UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Eléctrica

APLICACIÓN INFORMÁTICA PARA TELÉFONO MÓVIL PARA COMPLETAR UN BLOG

Memoria y Anexos

Autor: Andrea Díaz Zárate
Director: Javier Farreres de la Morena

Convocatoria: Mayo 2018

Resumen

La revolución digital está inmersa en nuestra existencia. Son tantos los avances sociales, científicos y tecnológicos en las vidas de las personas que el mundo necesita adaptarse a sus nuevas necesidades, gustos y costumbres.

La sociedad española es cada vez más viral, y utiliza su *smartphone* como complemento cuotidiano, tanto para situaciones sociales como profesionales. Esto deriva en un 92,9% de ciudadanos españoles que utiliza el dispositivo móvil todos los días para acceder a la red y en un 11% de incremento del uso de las apps.

Actualmente en nuestro país se encuentran alrededor de 27,7 millones de usuarios activos de aplicaciones que tienen instaladas una media de 17,8 apps en sus *smartphones*, según la información recogida en el informe *Distendia Mobile en España y en el Mundo 2017*.

En teléfonos móviles, el 86% de nuestro tiempo se dedica a aplicaciones y sólo el 14% a la web. Reflexionemos sobre nuestra actividad: cuando usamos nuestro teléfono ¿abrimos el navegador o una app?, ¿nos comunicamos a través de una web o de una app específica? El éxito de las aplicaciones está en que son más rápidas y más fáciles de utilizar que las páginas web.

A raíz de estos cambios y hábitos, decidí transformar el blog web gastronómico *Barneando* en una aplicación móvil para dispositivos Android. En este proyecto podéis encontrar las entrañas de este proceso: empezando por la historia teórica, el diseño y el desarrollo de la app, y acabando por la descarga en tu *smartphone*.

Resum

La revolució digital està immersa en la nostra existència. Són tants els avenços socials, científics i tecnològics en les vides de les persones que el món necessita adaptar-se a les seves noves necessitats, gustos i costums.

La societat espanyola és cada vegada més viral, i utilitza el seu telèfon intel·ligent com a complement quotidià, tant per a situacions socials com a professionals. Això deriva en un 92,9% ciutadans espanyols que utilitza el dispositiu mòbil cada dia per accedir a la xarxa i en un 11% d'increment de l'ús de les apps.

Actualment al nostre país es troben al voltant de 27,7 milions d'usuaris actius d'aplicacions que tenen instal·lades una mitjana de 17,8 apps en els seus *smartphones*, segons la informació recollida en l'informe *Distendia Mobile en España y en el Mundo 2017.*

En telèfons mòbils, el 86% del nostre temps es dedica a aplicacions i només el 14% a la web. Reflexionem sobre la nostra activitat: quan fem servir el nostre telèfon ¿obrim el navegador o una app?, ens comuniquem a través d'un lloc web o d'una app específica? L'èxit de les aplicacions està en que són més ràpides i més fàcils d'usar que les pàgines web.

Arran d'aquests canvis i hàbits, vaig decidir transformar el bloc web gastronòmic *Barneando* en una aplicació mòbil per a dispositius Android. En aquest projecte podeu trobar les entranyes d'aquest procés: començant per la història teòrica, el disseny i el desenvolupament de l'app, i acabant per la descàrrega d'aquesta al teu *smartphone*.

Abstract

The digital revolution is immersed in our existence. There are so many social, scientific and technological advances in people's lives that the world needs to adapt to their new needs, tastes and customs.

Spanish society is becoming more and more viral, and uses its smartphone as a daily complement, both for social and professional situations. This results in 92.9% of Spanish citizens using their mobile device every day to access the network and an 11% increase in the use of apps.

Currently in our country there are around 27.7 million active application users who have installed an average of 17.8 apps on their smartphones, according to the information contained in the report *Distendia Mobile en España y en el Mundo 2017*.

On mobile phones, 86% of our time in spent on applications and only 14% on the web. Let's think about our activity: when we use our smartphone, do we open the browser or an app? Do we communicate through a website or a specific app? The success of applications is due to the fact that they are faster and easier to use than web pages.

As a result of these changes and habits, I decided to transform the gastronomic web blog *Barneando* into a mobile application for Android devices. In this project you can find the entrails of this process: starting with the theoretical history, design and development of the app, and ending with the download to your smartphone.

Agradecimientos

En primer lugar, quiero dar las gracias a mi compañero de viaje, mi cómplice y amor Sergi. Gracias por tu energía, tus consejos y tu ayuda constante a lo largo de este proyecto.

En segundo lugar, a mi hermana Ainhoa y a mi padre por apoyarme siempre en todas y cada una de mis decisiones. Sin vosotros nada de esto habría sido posible.

Seguidamente agradecer también a Javier Farreres, mi tutor en este proyecto.

Y en último lugar agradecer a mis amigos, en especial a Aleix, Bekah y Viki por su ayuda y disposición en todo momento.

Índice

Índice de figuras

Índice de tablas

1. Prefacio

1.1. Origen del trabajo

La tecnología de los dispositivos móviles avanza a pasos agigantados y la aparición de aplicaciones móviles crece de manera exponencial. Todo aquel que disponga de un móvil con conexión a Internet y un Sistema Operativo adecuado tiene a su alcance una gran oferta de aplicaciones con todo tipo de funcionalidades y servicios.

La forma en que la gente interactúa entre sí y con las organizaciones se ha visto alterada con el auge de los *smartphones* y las tabletas, alzando nuevas oportunidades tanto para las empresas como para la sociedad.

Esta nueva etapa en la historia de las Tecnologías de la Información y la Comunicación (TIC) se caracteriza por dos grandes líderes de la industria de la movilidad. Por un lado, se sitúa el creador de los famosos iPhone y iPad, Apple, que incorporan el Sistema Operativo iOS; mientras que, por el otro se encuentra Google con su Sistema Operativo Android, el cual se mantiene como la plataforma más utilizada.

Esta popularidad de los dispositivos móviles está haciendo que los usuarios los utilicen para muchas de sus actividades, tanto personales como laborales, dejando a un lado el habitual ordenador personal.

Android es uno de los sistemas operativos que lideran el mercado de la tecnología de comunicación, y a consecuencia, existe una demanda muy exigente de un sector muy extenso de la población. Esta demanda se centra básicamente en aplicaciones que puedan mejorar el estilo de vida del usuario o incluso personalizar el propio dispositivo con el mismo fin.

1.2. Motivación

La motivación personal por la que he decidido realizar el trabajo de final de grado sobre el desarrollo de una aplicación móvil es porque se trata de uno de los sectores que ha tenido un mayor impacto en la sociedad y ha experimentado una mayor evolución en los últimos 10 años.

Los dispositivos móviles, como los *smartphones* (teléfonos inteligentes) o las tabletas, son los principales impulsores del cambio en la manera de relacionarse con personas, buscar cualquier información en un solo clic o simplemente poder ubicarnos en una ciudad desconocida gracias a la tecnología GPS.

Hoy en día, las aplicaciones móviles juegan un papel de gran relevancia ya que, en el periodo económico actual, protagonizado por gente joven y emprendedora, son muchos los que ven nuevas oportunidades en ellas. Existen aplicaciones de todo tipo, y cada una de ellas pretende cubrir las necesidades de sus usuarios.

Como punto final a los estudios de ingeniería, uno debe ser capaz de enfrentarse a cualquier reto que se le plantee, proponiendo y diseñando distintas alternativas con el fin de hallar la mejor solución.

En este proyecto, enfrentándome a un a un problema de diseño de software, he visto las puertas abiertas al futuro del sector de la ingeniería.

1.3. Objetivos

El objetivo principal de este presente trabajo es desarrollar una aplicación sencilla y funcional que ayude al usuario a encontrar sitios donde comer en Barcelona ciudad. Cualquier persona interesada debe ser capaz de utilizarla.

Los objetivos específicos para lograr el objetivo general de este proyecto son los siguientes:

- Obtener un conocimiento global sobre el lenguaje de programación Java.
- Obtener un conocimiento global sobre el sistema operativo Android. Para ello se estudiará en qué se fundamenta, cómo ejecuta sus aplicaciones, así como qué mecanismos y herramientas están disponibles para su desarrollo.
- Realizar una aplicación en la que se vea reflejado el blog gastronómico de *Barneando,* tanto en su contenido como en el diseño.
- Lograr realizar una aplicación móvil con una interfaz de usuario bonita y de fácil uso. Además de conseguir una aplicación útil y compacta.

1.4. Fases de realización

A continuación, se plantean las fases seguidas a lo largo de la realización de este proyecto.

- **Estudio:** Fase de aprendizaje de los lenguajes Java y Android.
- **Análisis:** Definición de los requisitos necesarios con el objetivo de resolver lo que se desea construir.
- **Diseño:** Tras finalizar las fases de estudio y análisis, y con los requisitos bien definidos, se procede a diseñar la arquitectura de la aplicación; diseño de pantallas, interfaz, seguridad, etc.
- **Implementación:** Programación de la aplicación en Android Studio.

- **Documentación:** Según una guía sobre desarrollo software, la documentación es algo indispensable para el correcto mantenimiento de cualquier producto software. Tanto a nivel de código como de documento, con el fin de dejar constancia por escrito de las fases llevadas a cabo en el ciclo de vida del producto software.
- **Mantenimiento:** Actualmente en este proyecto el mantenimiento puede no tener mucha cabida, pero se orienta a un trabajo futuro con bastante proyección con el fin de mejorar y ampliar la aplicación.

1.5. Descripción de la aplicación

Barneando, es una aplicación Android basada en el blog de restauración barneando.food.blog.

En muchas ocasiones, nos apetece variar e ir a algún sitio bonito, diferente y radiante donde ir a desayunar, comer, cenar, tomar un café, una cerveza, merendar, etc. rodeados de un ambiente espectacular.

En *Barneando* podrás descubrir nuevos lugares, restaurantes y bares de la ciudad de Barcelona donde disfrutar del mejor diseño de los locales gastronómicos de la ciudad.

Entre otras cosas ofrece:

- Tener al alcance de tu mano todas las entradas del blog
- Mediante la tecnología GPS, saber a qué distancia se encuentran de nosotros.
- Crear un ranking con tus restaurantes y bares favoritos.
- Llamar para reservar directamente desde la aplicación.

La aplicación de *Barneando* refleja el contenido del blog, ofreciendo al usuario un nuevo formato adaptado a cualquier terminal móvil y nuevas prestaciones que en el blog online no eran posibles.

2. Introducción a Java

Aunque existen otras opciones, la mayoría de aplicaciones Android se desarrollan utilizando el lenguaje Java, ya que tiene una serie de características que hacen que este lenguaje sea muy atractivo para el sistema Android.

Este lenguaje de programación, tiene como propósito posibilitar que los desarrolladores de aplicaciones puedan escribir el código una única vez y a su vez que puedan hacerlo correr en cualquier dispositivo, es decir, que el código creado en una plataforma cualquiera no tiene que ser recompilado para poder correr en otra (conocido en inglés como *WORA* o "*write once, run anywhere"*).

El diseño de Java, su sencilla portabilidad, su robustez y su apoyo de la industria, la han convertido en una de las lenguas con mayor crecimiento y que más se utilizan en las distintas áreas de la industria informática.

2.1. Características de Java

A continuación, se destacan las principales características que definen Java y lo diferencian de otros lenguajes de programación, haciéndolo fundamentalmente interesante para el desarrollo de aplicaciones.

- **Simple y familiar**

  Se puede decir que Java es un lenguaje familiar ya que está basado en el lenguaje C. Esta característica permite que millones de programadores que ya conocen este lenguaje se sientan mucho más cómodos.

  Al ser un lenguaje desarrollado desde cero, sin ninguna concesión de compatibilidad, proporciona la definición de un lenguaje muy sencillo y coherente y a la misma vez, sumamente potente.

  Aquellas características confusas o menos útiles del lenguaje C han sido eliminadas, como, por ejemplo, no existen los punteros, macros, tampoco definición de tipos, ni registros.

  Todas estas características hacen que sea un lenguaje de rápido aprendizaje, con menos errores (reducción de errores de hasta el 50%) y que el proceso de programación se simplifique notoriamente.

- **Programación Orientada a Objetos (POO)**

En el lenguaje Java está orientado a objetos desde la base. Gracias a esta técnica de programación se obtiene un producto de mejor calidad y a su vez se reduce de manera notoria el tiempo de desarrollo.

En Java todo son objetos (exceptuando los tipos elementales), es decir, cualquier definición de datos ha de ser un objeto, todas las funciones han de estar dentro de algún objeto y las librerías de funciones son también objetos.

Este lenguaje incorpora las técnicas más importantes de la programación orientada a objetos:

- Encapsulación: Consiste en el hecho de ocultar el estado o los datos miembro de un objeto, de tal forma que solamente es posible modificarlos mediante los métodos definidos para dicho objeto.
- Herencia: Permite definir una clase tomando como base otra clase ya existente. A consecuencia de heredar de una clase base, también heredaremos los atributos y los métodos, en cambio, los constructores no se heredan, pero son utilizados.

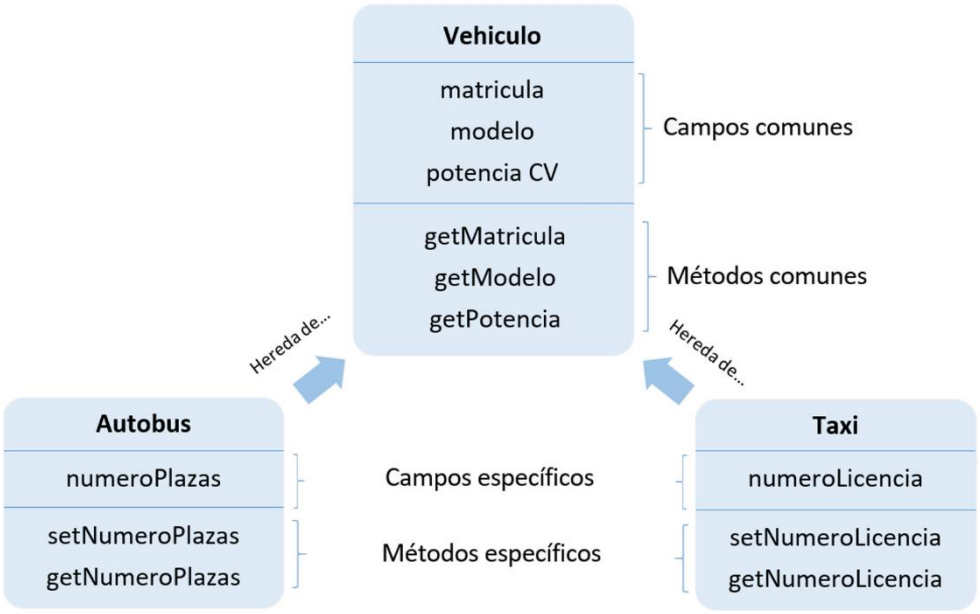


Figura 2.1 Ejemplo de la característica herencia (fuente propia)

- Polimorfismo: Un objeto es considerado de la clase a la que pertenece y también de las clases de las cuales desciende. Podemos trabajar con objetos como si fueran de la clase vehículo, aunque sean de una clase más específica, como autobús o taxi.

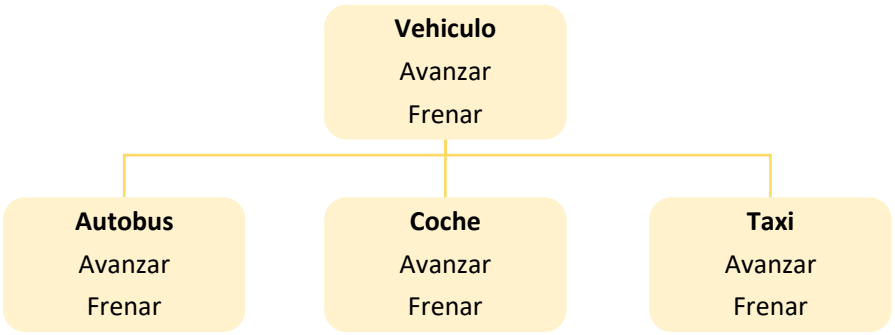


Figura 2.2 Ejemplo de la característica polimorfismo (fuente propia)

- **Independiente de la plataforma**

El objetivo primordial con el que se creó Java, era que el programa pudiera ser ejecutado en cualquier procesador o sistema operativo, independientemente de la plataforma. Para ello se definió una nueva arquitectura independiente de las que ya existían, es decir, totalmente neutra.

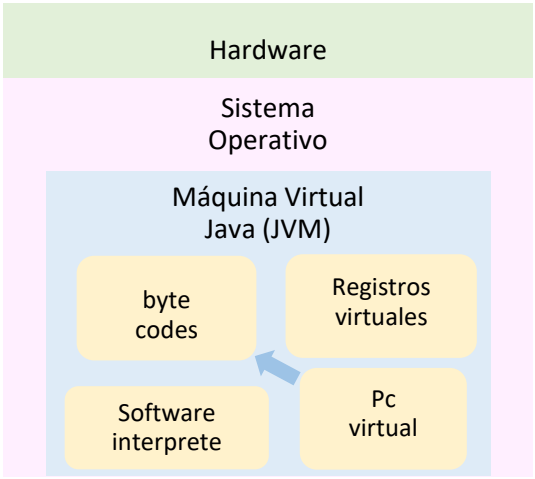


Figura 2.3 Arquitectura neutra de Java (fuente propia)

Al compilar un programa en Java, se generan unas instrucciones que son destinadas a una máquina virtual (conocidas como *bytecodes*). En cuanto queramos ejecutar el programa en un procesador, la máquina virtual tendrá que ser emulada. De esta manera, al tener todas las máquinas virtuales la misma arquitectura, se solucionan todos los problemas de portabilidad. De la misma forma, también se soluciona el problema de presentación (codificación de datos), ya que el programa en Java cree que siempre se está ejecutando en la misma máquina.

- **Interpretado**

Para poder ejecutar cualquier programa en Java, es necesario simular una máquina virtual ya que los programas no pueden ser ejecutados directamente por ningún compilador.

Las instrucciones destinadas a la máquina virtual (*bytecodes*) han de ser interpretados, sin embargo, el programa Java no sigue el esquema típico de un programa interpretado (escritura / compilación / interpretación).

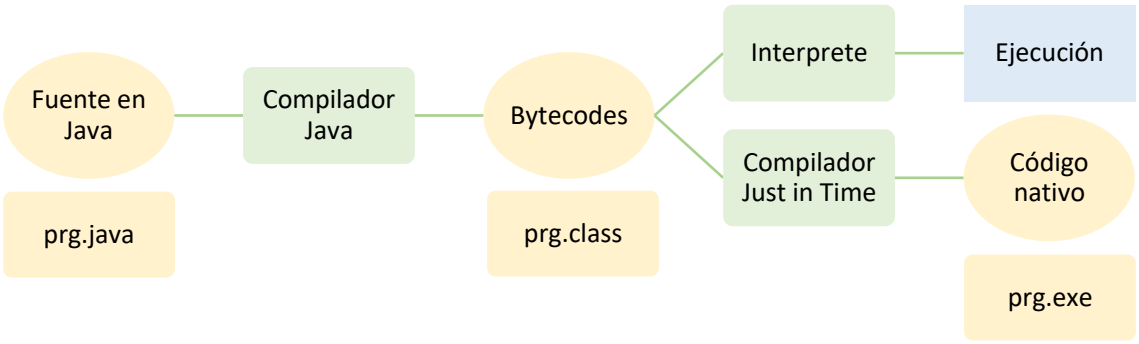


Figura 2.4 Esquema del programa Java (fuente propia)

Al ser un programa interpretado, su ejecución es poco eficiente. Si en algún momento se quiere mejorar el tiempo de ejecución del programa, se puede compilar en *Just in Time*, es decir, compilar los *bytecodes* en código nativo de la plataforma.

- **Seguro**

Hoy en día la seguridad resulta crucial, y más en el desarrollo de aplicaciones distribuidas. Es por eso que Java extrema las medidas de seguridad.

Para ello se ha de garantizar al consumidor que cuando se descarga un programa de la red, este no pueda contener un virus, curiosear en la máquina local o destruir datos.

Java es especialmente seguro. Primeramente, soporta la seguridad "*sandboxing"* o ejecución en caja de arena, dado que la máquina virtual es realmente un software, ningún programa es capaz de tomar el control del procesador. Esto nos permite aislar totalmente lo que es la máquina virtual del entorno real.

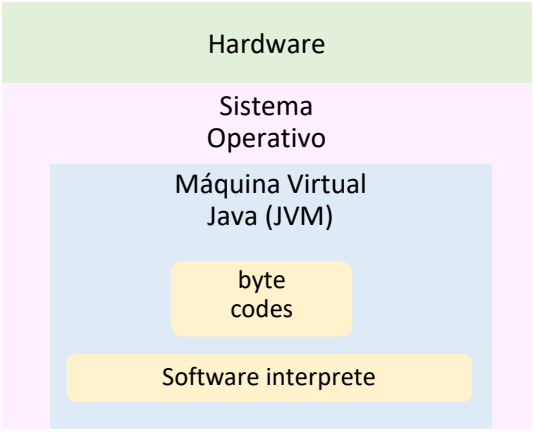


Figura 2.5 Seguridad "sandboxing" o caja de arena (fuente propia)

Además, un compilador Java asegura que el código fuente no infrinja las normas de seguridad y también problema los *bytecodes* son verificados antes de su ejecución.

- **Robusto**

Una de las ventajas de Java es que al ejecutarse dentro de una máquina virtual es muy difícil que un programa en Java bloquee el sistema, eso lo hace además de seguro, un lenguaje robusto.

Asimismo, la asignación de tipos es muy estricta, es decir, no se pueden cometer este tipo de errores.

En cuanto a la gestión de la memoria, esta es llevada a cabo directamente por el sistema, es decir, el consumidor no tiene que reservar ningún tipo de memoria cosa que suele causar muchos problemas de la programación actual.

Finalmente, Java realiza un chequeo del código tanto en tiempo de ejecución como de compilación.

- **Distribuido y dinámico**

Java ha sido diseñado para una ejecución remota y distribuida, es decir todas las clases (ficheros *.class*) pueden enlazarse dinámicamente y transportado a través de la red. Estas clases son cargadas dinámicamente en tiempo de ejecución, en el momento que se necesitan.

Además, es un lenguaje dinámicamente extensible, diseñado para poder adaptarse a entornos en constante evolución.

- ***Multi-thread***

Un programa puede lanzar más de un hilo de ejecución (proceso individual ejecutándose en un sistema) o *thread*, de forma que no se crean nuevos procesos, sino que cada *thread* va a compartir el código y las variables con el *thread* principal. Es decir, cuando hablamos de *multi-thread* (multihilo), se refiere a que dos o más tareas se ejecutan a la vez dentro de un mismo programa.

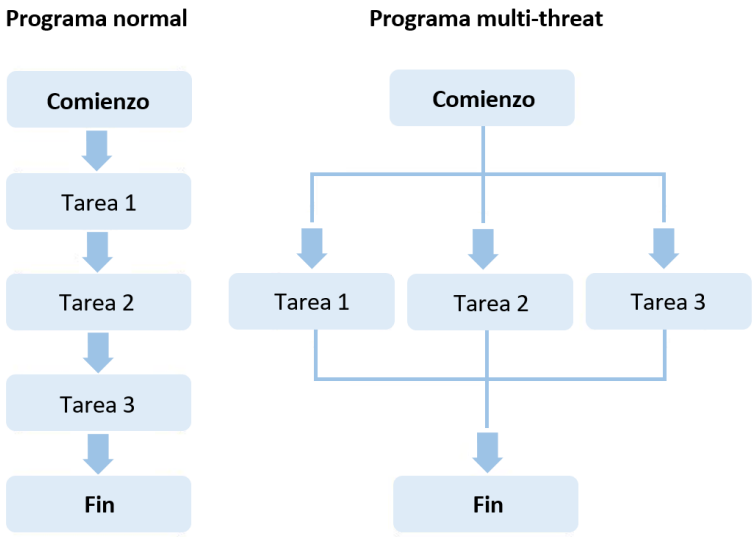


Figura 2.6 Diferencias entre un programa normal y un programa multi-thread (fuente propia)

De esta manera, simultáneamente se pueden atender a varias tareas de forma sencilla y sin tener que crear nuevos procesos, es una solución sencilla y elegante a la multiprogramación.

2.1. Clases en Java

Como se ha mencionado anteriormente, Java es un lenguaje orientado a objetos (POO), por lo tanto, resulta imprescindible saber cómo definir clases en Java.

- **Objeto:** Es una entidad existente en la memoria del ordenador y que dispone de unas propiedades (datos sobre sí mismo almacenados por el objeto o **atributos**) y un comportamiento (**métodos**). Se puede decir que un objeto es una instancia de una clase.
- **Clase:** Define un tipo de objeto detallando que propiedades (atributos) y operaciones disponibles va a tener.

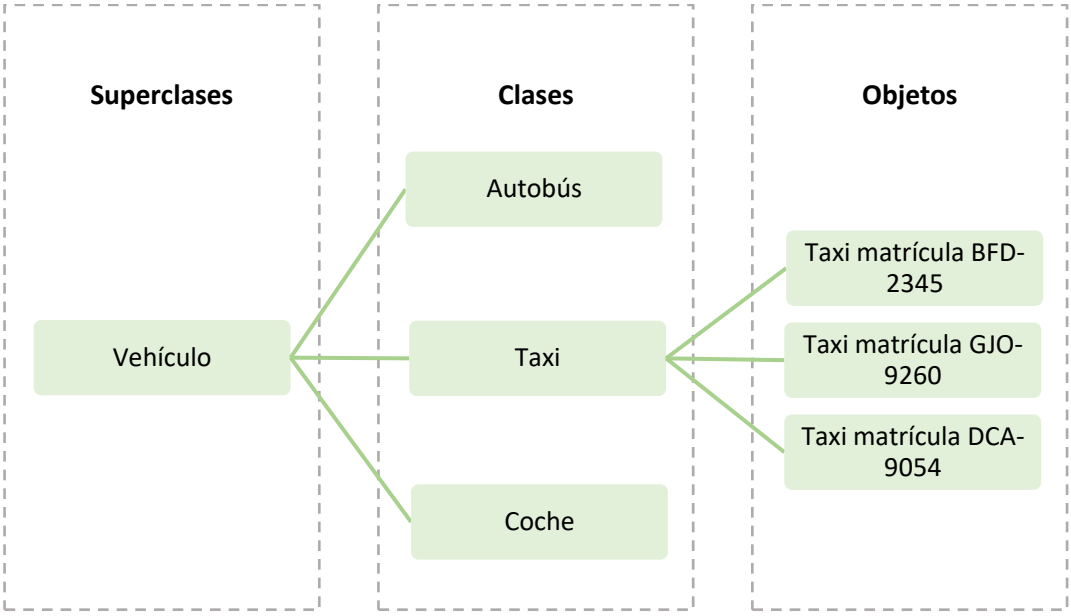


Figura 2.7 Ejemplo de superclases, clases y objetos (fuente propia)

En el ejemplo anterior hemos considerado que nuestro programa trata de gestionar datos sobre los vehículos en una ciudad. Para poder gestionar los vehículos, creamos la superclase (clase que tiene todos los atributos y métodos en común), vehículo, las clases: autobús, taxi y coche, y luego los objetos de cada clase.

Una clase está formada por:

- **Atributos:** Conocidos también campos o propiedades, son aquellos que almacenan alguna información del objeto. Definen su estado.
- **Constructor(es):** Son llamados igual que la clase y son utilizados para inicializar las variables de un objeto.
- **Métodos:** Son utilizados para consultar o cambiar el estado de un objeto.

```
class <Clase>  {
  //declaración de atributos
  [visibilidad] [modificadores] <tipo> <atributo>  [=valor];
  ...
    //declaración de constructor
  public <Clase>(<argumentos>)  {
     <instrucciones>;
  }
    //declaración de métodos
  [visibilidad] [modificadores]  <tipo> <método>(<argumentos>)  {
     <instrucciones>;
  }
  ...
}
```

Figura 2.8 Estructura de una clase

A lo que se refiere a estructura de una clase, lo habitual es comenzar declarando los atributos. Cada atributo puede tener una visibilidad, una serie de modificadores y un tipo de atributo. El tipo de atributo puede ser una clase de objeto definido previamente o un tipo simple.

Tipo	Tamaño	Descripción	Rango	
Números enteros				
Byte	8 bits	Entero de un byte	-128	127
Short	16 bits	Entero corto	-32.768	32.767
Int	32 bits	Entero	-2.147.483.648	2.147.483.647
Long	64 bits	Entero largo	$-9.223.372 \cdot 10^{12}$	$9.223.372 \cdot 10^{12}$
Números reales				
Float	32 bits	Coma flotante de precisión simple	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$
Doublé	64 bits	Coma flotante de precisión doble	$-1.8 \cdot 10^{308}$	$1.8 \cdot 10^{308}$
Otros				
Char	16 bits	Unicode (un sólo carácter)	Unicode 0	Unicode 2^{16} -1
boolean	true / false	Un valor booleano (verdadero o falso)	False	True

Tabla 2.1 Tipos simples en Java (fuente propia)

En los atributos se almacena la información que una clase necesita y que define al objeto en cuestión.

Seguidamente se declara el constructor, que es el método que utiliza el mismo nombre que la clase y que se utiliza para poder inicializar el objeto.

Finalmente, se declaran los métodos. Estos son las funciones o los procedimientos de otros lenguajes de programación y se utilizan para consultar o modificar el estado del objeto. Los métodos suelen empezar mostrando su visibilidad (*public.protected* o *private*).

2.2. Encapsulamiento y visibilidad en *Java*

Cuando se diseña un software hay dos matices que son fundamentales, la interfaz y la implementación.

En java la **interfaz** (*interface*) se define como todos los componentes de una clase que son visibles desde el exterior de esta (con visibilidad *public*). En esta se especifica que se debe hacer, pero no su implementación.

En cambio, la **implementación** es el código exacto utilizado para implementar los diferentes algoritmos y los diferentes atributos que se utilizan en la implementación de la clase.

El termino encapsulamiento se basa en ocultar los atributos y a la implementación de un objeto, de tal forma que la única manera de cambiar su estado es a través de ciertas operaciones descritas en la interfaz del objeto. Lo que se intenta, es que la interfaz sea totalmente independiente de la implementación de esta clase.

En la programación en Java, el encapsulamiento está fuertemente enlazado con la visibilidad. Para poder indicar la visibilidad de un elemento (estos elementos pueden ser atributos o métodos) se añaden las siguientes palabras reservadas:

- ***public*:** El atributo o el método va a poder ser accesible desde cualquier clase, por tanto, forma parte de la interfaz de nuestra clase.
- ***private*:** Sólo se puede acceder desde la clase actual.
- ***protected*:** Término medio donde sólo se puede acceder desde la clase actual, sus descendientes y clases del mismo paquete.
- **<no indicar nada>:** En este caso sólo es accesible desde cualquier clase de nuestro paquete.

Los atributos de una clase están fuertemente relacionados con su implementación y por lo tanto es conveniente marcarlos siempre como "*private*" para poder impedir cualquier acceso desde fuera.

Además, una de las mayores ventajas de ocultar los atributos es que esto hace que sea posible cambiarlos en cualquier momento sin que afecte a la interfaz.

3. Introducción a la API de Android

Android es un sistema operativo que se basa en el núcleo Linux. Principalmente fue diseñado para dispositivos móviles que poseen de pantalla táctil, como, por ejemplo, teléfonos inteligentes (*smartphones*), tabletas, relojes inteligentes, automóviles relojes inteligentes (*smartwatches*), etc.

3.1. Características de Android

Hoy en día, existen muchísimas plataformas para móviles (Apple iOS, Windows Phone, Linux Mobile, etc.), no obstante, Android muestra una serie de características que lo hacen especial:

- **Plataforma realmente abierta:** Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus mayores ventajas es que se puede utilizar y personalizar el sistema sin pagar *royalties*.

- **Adaptable a cualquier tipo de hardware**: La plataforma Android no ha sido diseñada únicamente para su uso en teléfonos móviles y tabletas. A día de hoy, ya se pueden encontrar relojes, cámaras, electrodomésticos y gran variedad de sistemas empotrados que se basan en este sistema operativo. Esto supone enormes ventajas, aunque va a suponer un trabajo extra al programador. Las aplicaciones han de funcionar de manera correcta en dispositivos con gran diversidad de tipos de entrada, pantalla, memoria… Esta propiedad difiere totalmente con la estrategia de Apple. En iOS se han de desarrollar aplicaciones diferentes para cada uno de sus dispositivos (iPhone, iWatch, iPad…).

- **Portabilidad asegurada**: Al ser las aplicaciones finales desarrolladas en Java nos garantiza que en todo momento podrán ser ejecutadas en cualquier tipo de CPU. Esto es gracias al concepto de máquina virtual.

- **Arquitectura basada en componentes inspirados en Internet**: Al realizarse el diseño de la interfaz de usuario en XML, posibilita que una misma aplicación se ejecute en un reloj de pantalla reducida, en un televisor o en un dispositivo de telefonía móvil.

- **Filosofía de dispositivo siempre conectado a Internet**. Hoy en día, la mayoría de aplicaciones solamente funcionan si se dispone de una conexión constante a Internet.

- **Gran cantidad de servicios incorporados**. Son muchísimos los servicios que se pueden incorporar hoy en día; multimedia, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, huella dactilar, etc.

- **Aceptable nivel de seguridad**. Como se ha explicado anteriormente la seguridad hoy en día es un tema primordial. Gracias al concepto de ejecución dentro de una caja de arena o *sandboxing* (explicado anteriormente en el punto 2.1.5.), los programas se encuentran totalmente aislados unos de otros. Además, cada aplicación establece una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, acceso a contactos, etc.). Además, desde la versión 6.0 el usuario puede conceder o retirar permisos a las aplicaciones en todo momento desde los ajustes del dispositivo.

- **Optimizado para baja potencia y poca memoria**. A la hora de diseñar la plataforma Android se ha considerado específicamente el *hardware* de los dispositivos móviles. Google ha implementado la máquina virtual de Java optimizándola para el uso en dispositivos móviles. Android utiliza esta nueva implementación llamada Máquina Virtual ART (o Dalvik en versiones anteriores).

- **Alta calidad de gráficos y sonido**. La plataforma incorpora gráficos vectoriales suavizados, animaciones entre otras cosas. También soporta los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

Android combina características muy atrayentes, no obstante, toda gran empresa tiene su gran rival, en su caso es el gigante de la manzana Apple.

A día de hoy, Android sigue siendo el sistema operativo dominante, aunque ambas plataformas siguen luchando y mejorando constantemente para ganar el podio.

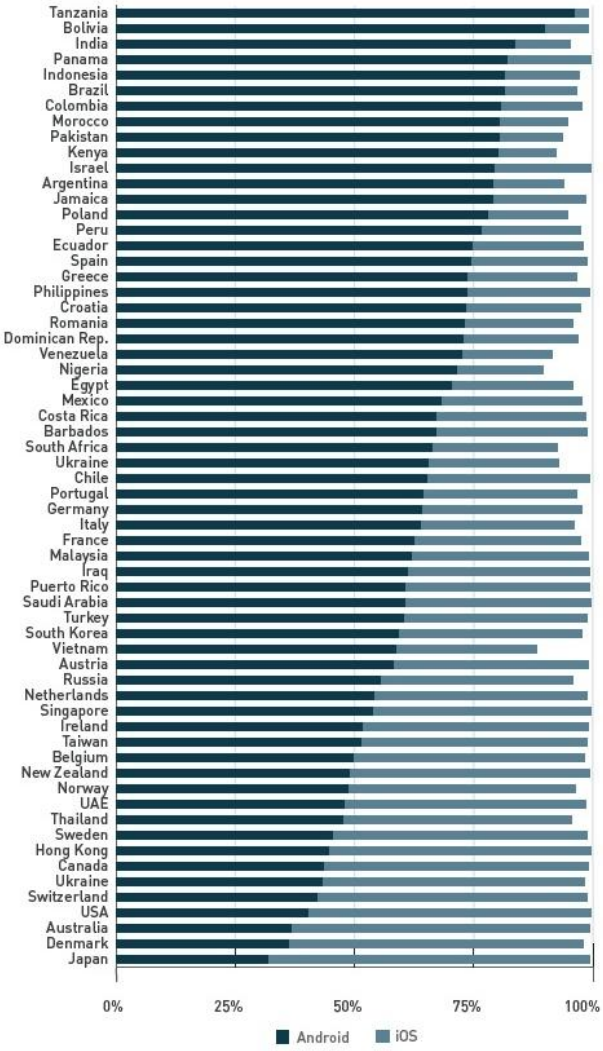


Figura 3.1 Comparación entre el mercado de Android y iOS (Fuente: DeviceAtlas Mobile Web Intelligence Report August 2017)

3.2. Orígenes de Android

A finales del año 2003, en la localidad de Palo Alto, California, Andy Rubin, Rich Miner, Chris White y Nick Sears fundan la compañía Android Inc. con el objetivo de desarrollar un sistema operativo para sistemas móviles basado en Linux.

La gran multinacional Google adquiere Android Inc. en el año 2005. En este mismo año se empieza a trabajar en la creación de la máquina virtual Java optimizada para terminales móviles (Dalvik VM).

El 5 de noviembre de 2007 se crea la *Open Handset Alliance,* un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio, con el objetivo de desarrollar estándares abiertos para telefonía móvil. Este conglomerado está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel

entre otros. Los miembros de este conglomerado se comprometen a publicar parte de su propiedad intelectual como código abierto bajo licencia Apache v2.0.

El mismo día se anuncia la primera versión del sistema operativo: Android 1.0 Apple Pie y se lanza una primera versión del Android SDK.

En el año 2008, aparece el primer móvil con sistema operativo Android (T-Mobile G1). En octubre de ese mismo año, Google libera el código fuente de Android. En el mismo mes, se abre Android Market, para la descarga de aplicaciones.

A mediados del año 2009, Android lanza la versión 1.5 del *Android SDK*, incorporando nuevas características, como el teclado en pantalla. A finales de este mismo año, se lanza la versión 2.0 y durante el año siguiente las versiones 2.1, 2.2 y 2.3.

En el año 2010, Android ya es considerado uno de los sistemas operativos más utilizados a nivel mundial, con resultados muy cercanos a los de su mayor competidor iOS, e incluso superando al sistema de Apple en Estados Unidos en el segundo y tercer trimestre con una cuota de mercado en el tercer trimestre del 43,6%.

Al año siguiente, se lanza la versión 3.x (Honeycomb) para tabletas y la 4.0 (Ice Cream Sandwich), para teléfonos móviles y tabletas. En este mismo año, Android se consolida como la plataforma número uno para móviles, alcanzando una cuota de mercado superior al 50% (más del doble que el segundo sistema operativo, iOS de Apple, Inc.).

Desde entonces Android no ha parado de evolucionar. Actualmente Android mantiene su espectacular crecimiento y a finales de este mismo año obtuvo una cuota de mercado de más del 80% a nivel mundial.

Figura 3.2 Cuota de mercado mundial por sistema operativo para el año 2016 (fuente: statista.com)

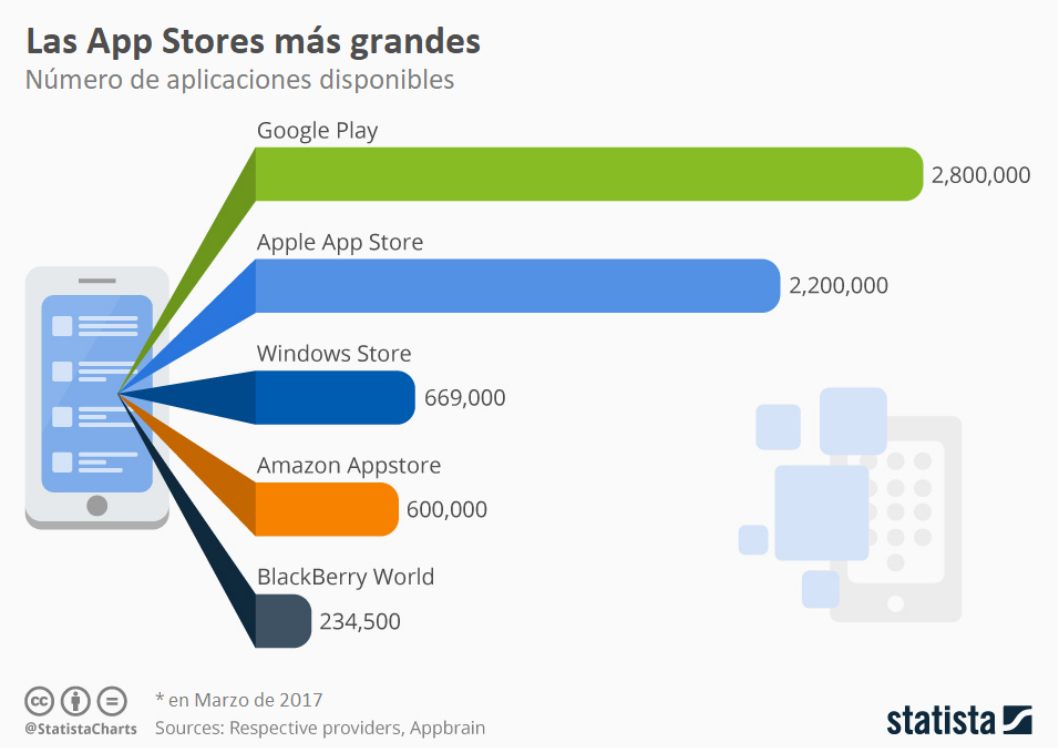


Figura 3.3 App Stores más grandes para el año 2017 (fuente: statista.com)

3.3. Arquitectura de Android

El siguiente grafico muestra una visión global por capas de cuál es la arquitectura empleada en Android. Como se puede observar está formada por cuatro capas diferentes. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece los suyos a capas superiores. Una de las características con más importancia, es que todas las capas están basadas en software libre.

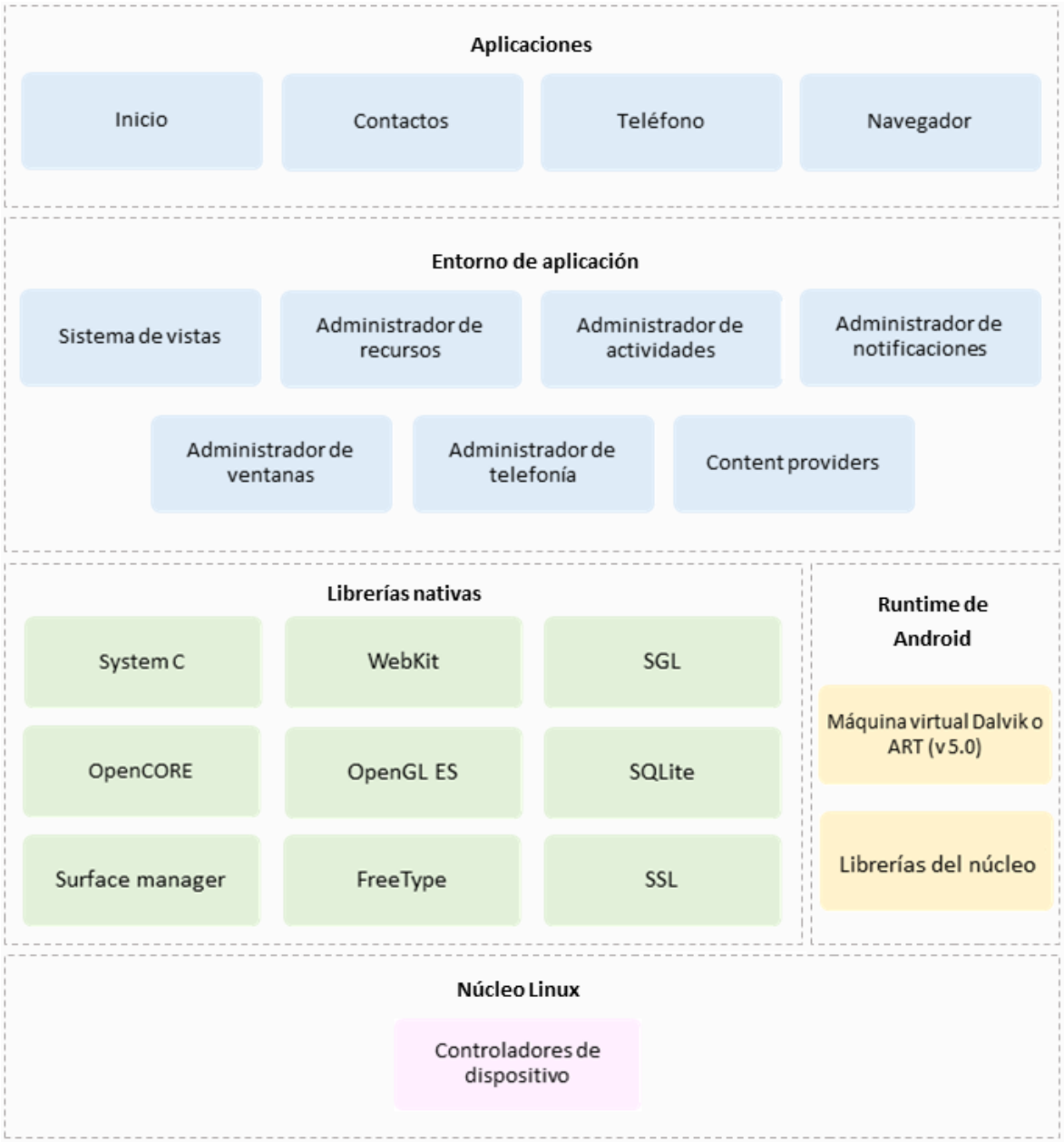


Figura 3.4 Arquitectura de Android (fuente propia)

- **El núcleo Linux**

Android utiliza el núcleo Linux 2.6 como base de la plataforma. Esta capa es utilizada como una capa de abstracción entre el hardware disponible en los dispositivos móviles y el resto de la pila. Por lo tanto, es la única capa dependiente del *hardware*.

Proporciona servicios como la seguridad, la administración de memoria de bajo nivel, la generación de multiprocesos, la pila de protocolos y el soporte de *drivers* para dispositivos. El uso del *kernel* (o núcleo) de Linux permite aprovechar las funciones de seguridad permitiendo al mismo tiempo que los fabricantes de dispositivos desarrollen controladores de *hardware* para un *kernel* conocido.

- ***Runtime* de Android (tiempo de ejecución)**

Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución o *Runtime* de Android. Está basado en el concepto de la máquina virtual utilizada en Java, pero dadas las limitaciones de memoria y procesador (dispositivos móviles), Google decidió crear una nueva, la máquina virtual *Dalvik*.

A partir de Android 5.0 (nivel de API 21), se reemplaza *Dalvik* por ART. Esta nueva máquina virtual consigue ejecutar varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos *Dalvik* ejecutables (.dex), un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo. Gracias a esto se consigue reducir el tiempo de ejecución del código Java hasta un 33%.

En el *Runtime* de Android también se incluye el módulo *Core Libraries* (librerías del núcleo), con multitud de librerías disponibles en el lenguaje Java.

- **Librerías nativas**

La siguiente capa incluye un conjunto de librerías utilizadas por Android. Estas librerías están escritas en C y C++, ya que están compiladas en código nativo del procesador. Junto al núcleo basado en Linux, estas librerías forman el corazón de Android.

Entre las librerías más importantes se pueden encontrar las siguientes:

  - **System C library:** Es una derivación de la librería BSD de C estándar (*libc*), adaptada para dispositivos empotados basados en Linux. Incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen también en este lenguaje.

- **Media Framework:** Librería que proporciona todos los códecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes tanto estáticas como animadas, MPEG4, H.264, MP3, AAC, AMR, JPG y PNG. Está basada en *OpenCORE* de *PacketVideo*.
- **Surface Manager:** Es la librería que maneja el acceso al subsistema de representación gráfica en 2D y 3D. Es la que se ocupa de componer los diferentes elementos navegación en pantalla.
- **WebKit/Chromium:** Es la librería que suministra un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android y en la vista Webview. En la versión 4.4, WebKit ha se reemplaza por Chromium/Blink, que forma la base del navegador Chrome de Google.
- **SGL:** Proporciona el motor de gráficos 2D, por tanto, será la librería más utilizada por la mayoría de aplicaciones.
- **Librerías 3D:** Junto con la librería SGL, representan las librerías graficas. Maneja gráficos en 3D y está basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D. Una característica muy importante de la capacidad grafica que tiene Android, es que se pueden desarrollar aplicaciones que combinen gráficos 2D y 3D.
- **FreeType:** Librería que permite trabajar rápidamente y sencillamente con diferentes fuentes en bitmap y renderizado vectorial.
- **SQLite:** Librería que crea y gestiona las bases de datos relacionales disponibles para todas las aplicaciones.
- **SSL:** Posibilita servicios de encriptación *Secure Socket Layer* (capa de conexión segura) para establecer comunicaciones seguras.

- **Entorno de aplicación**

Esta capa representa fundamentalmente una plataforma de desarrollo libre para cualquier aplicación. Ha sido diseñada para la posible reutilización de componentes, de esta misma forma, toda aplicación que está desarrollada para Android, ya sean las propias aplicaciones del dispositivo, las desarrolladas por el propio usuario o las que son creadas por diferentes compañías, utilizan el mismo API y el mismo *framework*, todo ello representado por este nivel.

Las API más importantes que incluye son:

- **Sistema de vistas (*View Manager*):** Proporciona un extenso conjunto de vistas (parte visual de los componentes), y abundantes elementos para la construcción de interfaces de usuario, como, por ejemplo, botones, lista, tamaño de ventanas, etc.

- **Administrador de recursos (*Resource Manager*):** Provee acceso a recursos que no son en código.
- **Administrador de actividades (*Activity Manager*):** Conjunto de API que maneja el ciclo de vida de las aplicaciones y facilita un sistema de navegación entre ellas.
- **Administrador de notificaciones (*Notification Manager*):** Autoriza a las aplicaciones, mediante un mismo formato, comunicarse con el usuario eventos que ocurren mediante su ejecución mostrando alertas personalizadas en la barra de estado.
- **Content providers:** Permite a cualquier aplicación compartir sus datos con otras aplicaciones de Android.
- **Administrador de ventanas (*Window Manager*):** Gestiona las diferentes ventanas de las aplicaciones utilizando la librería explicada anteriormente, Surface Manager.
- **Administrador de telefonía (*Telephone Manager*):** Contiene el conjunto de API relacionadas a las funciones propias del teléfono.

- **Aplicaciones**

Este nivel contiene el conjunto de aplicaciones, incluidas por defecto de Android como aquellas que el usuario añade posteriormente, instaladas en la máquina Android. Todas y cada una de estas aplicaciones utilizan los servicios, las API y las librerías de los niveles anteriores. Además, han de ejecutarse en la máquina virtual *Dalvik* (o ART a partir de la versión 5.0), para poder garantizar la seguridad del sistema.

Habitualmente las aplicaciones en sistema Android están escritas en Java, y para ello, normalmente se utiliza el Android SDK (*Software Development Kit*). Sin embargo, existe otra manera de desarrollar estas aplicaciones, y es usando C o C++. En este caso, comúnmente se utiliza el Android NDK (*Native Development Kit*).

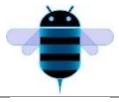
3.4. Versiones de Android y niveles de API

El primer paso que hay que realizar antes de empezar, es escoger la versión del sistema para la que se desea realizar la aplicación. Es importante saber que hay métodos y clases que solamente se encuentran disponibles a partir de una versión determinada, por eso mismo, se ha de saber con anterioridad que es lo que se va a usar, para poder conocer la versión mínima de API necesaria.

La interfaz de programación de aplicaciones, conocida como API (*Application Programming Interface*), es una agrupación de subrutinas, funciones y procedimientos (o métodos, en el caso de la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Cada vez que se lanza una nueva versión de Android, esta es compatible con todas y cada una de las versiones que se han lanzado anteriormente, es decir, solo se van incorporando nuevas funcionalidades.

Las versiones de Android se pueden identificar de tres maneras diferentes: A través del número versión, el nivel de API y por el nombre comercial. El nivel de API siempre son números enteros, comenzando por el numero 1. Para el nombre comercial se han utilizado nombres de postres siguiendo un orden alfabético: Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat y Oreo lanzado el pasado 21 agosto de 2017.

Nombre comercial	Número de versión	Fecha de lanzamiento	Nivel de API
Apple Pie 	1.0	23 de septiembre de 2008	1
Banana Bread 	1.1	9 de febrero de 2009	2
Cupcake 	1.5	25 de abril de 2009	3
Donut 	1.6	15 de septiembre de 2009	4
Eclair 	2.0–2.1	26 de octubre de 2009	5-7
Froyo 	2.2–2.2.3	20 de mayo de 2010	8
Gingerbread 	2.3–2.3.7	6 de diciembre de 2010	9–10
Honeycomb 	3.0–3.2.6	22 de febrero de 2011	11–13
Ice Cream Sandwich 	4.0–4.0.5	18 de octubre de 2011	14–15
Jelly Bean 	4.1–4.3.1	9 de julio de 2012	16–18
KitKat 	4.4–4.4.4, 4.4W–4.4W.2	31 de octubre de 2013	19–20

Lollipop 	5.0–5.1.1	12 de noviembre de 2014	21–22
Marshmallow 	6.0–6.0.1	5 de octubre de 2015	23
Nougat 	7.0 - 7.1 - 7.1.1 - 7.1.2	15 de junio de 2016	24-25
Oreo 	8.0 - 8.1	21 de agosto de 2017	26-27

Tabla 3.1 Plataformas Android lanzadas hasta la fecha (fuente propia)

3.5. Elegir versión en una aplicación Android

Lo primero que se ha de realizar antes de empezar un proyecto en Android, es escoger la versión del sistema. Es importante saber que hay clases y métodos que solo estarán disponibles a partir de una versión determinada, es decir, si se van a utilizar hay que conocer la versión mínima necesaria.

Como anteriormente se ha mencionado, las plataformas nuevas siempre son compatibles con las anteriores, ya que solo se añaden nuevas funcionalidades y en caso de modificar alguna funcionalidad no se elimina, se etiquetan como obsoletas, pero se pueden continuar utilizando.

Versión	Nombre	API	Distribución
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.5%
5.0	Lollipop	21	4.9%
5.1		22	18.0%
6.0	Marshmallow	23	26.0%
7.0	Nougat	24	23.0%
7.1		25	7.8%
8.0	Oreo	26	4.1%
8.1		27	0.5%

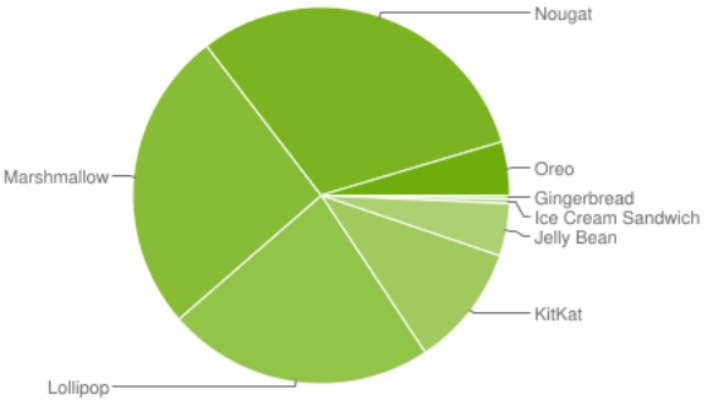


Figura 3.5 Dispositivos Android, según la plataforma instalada, que han accedido a Google Play Store el 16 de abril de 2018 y los 7 días anteriores. Las versiones con porcentajes inferiores al 0,1% no se muestran (fuente: developer.android.com)

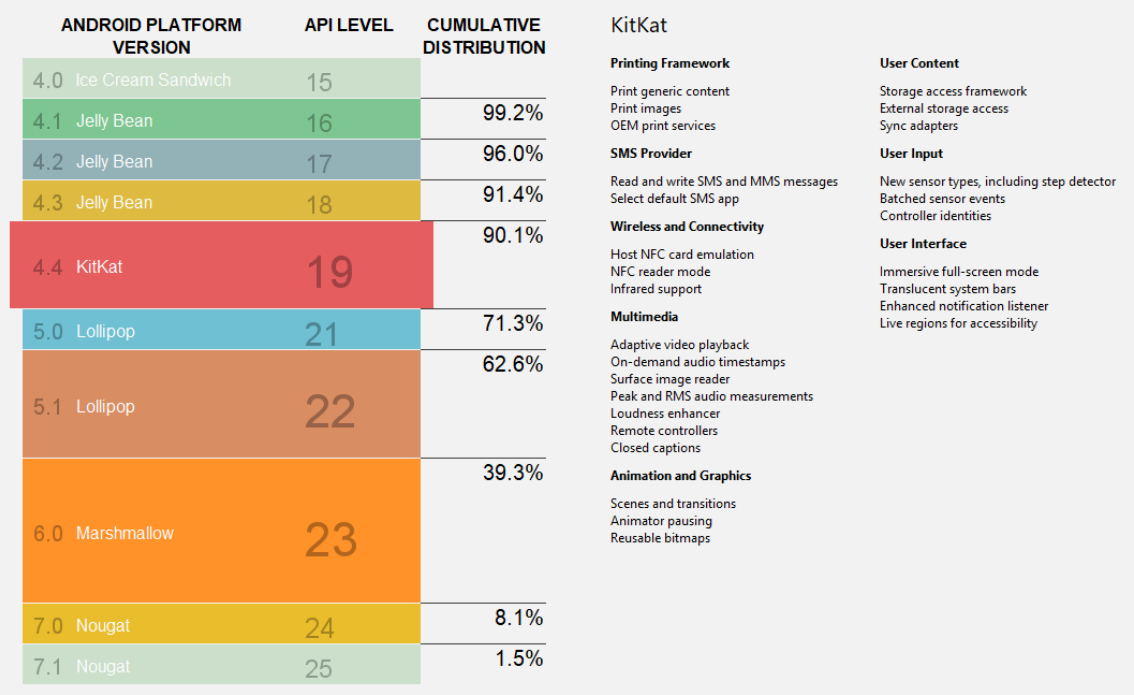


Figura 3.6 Distribución acumulativa de las versiones de Android (fuente: Android Studio)

Después de estudiar la gráfica anterior, se puede destacar el reducido número de usuarios que utilizan la versión Jelly Bean (menos de un 5%). Por lo tanto, puede ser buena opción utilizar como versión mínima la 4.4 (API 19) para el desarrollo de la aplicación, ya que con ella se cubre a más del 90% de los terminales.

3.6. Desarrollo de aplicaciones con Android Studio

Para un correcto desarrollo de aplicaciones con la herramienta Android Studio se han de seguir los pasos siguientes:

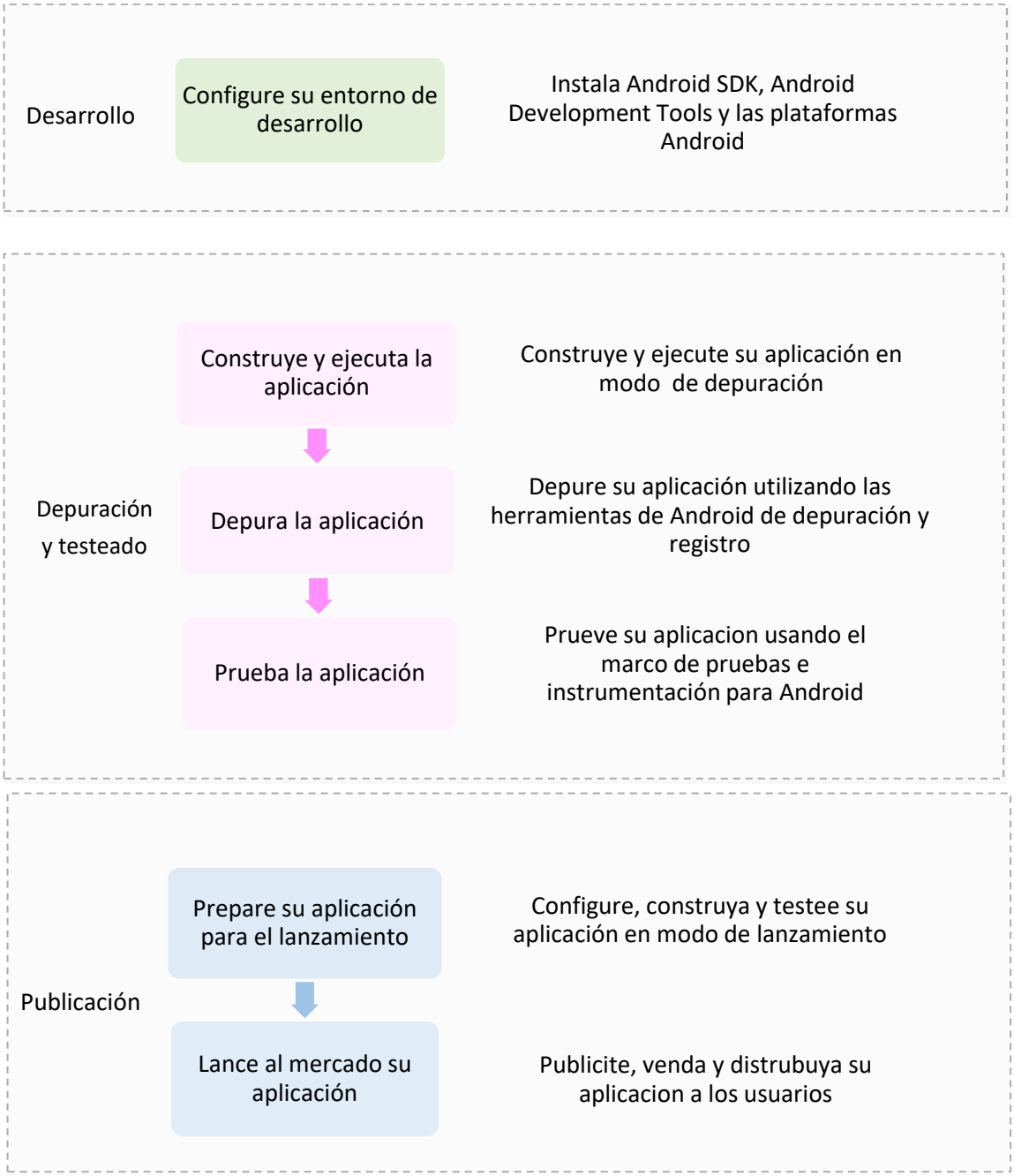


Figura 3.7 Ciclo de desarrollo de aplicaciones en Android Studio (fuente propia)

Una vez que la aplicación funciona, se puede lanzar al mercado. Para ello Android ha creado un mercado llamado Google Play donde los desarrolladores cuelgan sus aplicaciones para que todo usuario pueda descargárselas de forma fácil, rápida y segura.

4. Características del lenguaje UML

El lenguaje unificado de modelado o más conocido por sus siglas UML, tiene como objetivo crear un lenguaje de modelado visual común y sintácticamente elaborado para la arquitectura, el diseño y la implementación de sistemas de software complejos.

Este lenguaje es comparable a los planos usados en otros campos tecnológicos y consiste en diferentes tipos de diagramas que en general, describen la estructura, los límites y el comportamiento del sistema.

UML no es considerado un lenguaje de programación en sí, aunque se hallan diferentes herramientas que se pueden emplear para poder crear código en diferentes lenguajes usando estos diagramas.

Además, el lenguaje unificado de modelado tiene un vínculo directo con el análisis, el diseño y la programación orientada a objetos.

La finalidad principal de estos diagramas es presentar diferentes puntos de vista de un sistema.

A continuación, se describirán los diagramas que se han utilizado en esta misma memoria:

- **Diagrama de Casos de Uso**

  Un caso de uso es una descripción de las acciones que contiene un sistema desde el punto de vista del usuario.

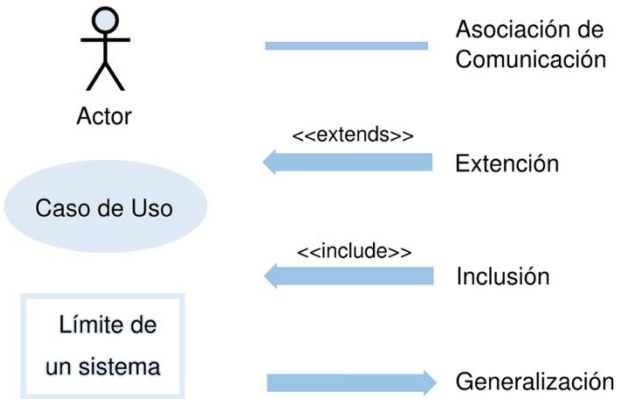


Figura 4.1 Notación de caso de uso (fuente propia)

Como se puede observar en la figura anterior, el rectángulo representa los límites del sistema que contiene los casos de uso. Los casos de uso se representan en forma de ovalo y son etiquetados

con la función del sistema. Finalmente, los actores son los usuarios del sistema y siempre estarán ubicados fuera de los límites de este.

Las asociaciones de comunicación, es decir, las relaciones entre un actor y un caso de uso, se representan con una línea simple. Para representar asociaciones entre clases de uso, se utilizan flechas etiquetadas con <<*include*>> o <<*extends*>>. Una relación "incluir" o <<*include*>>, indica que un caso de uso es necesario por otro para poder realizar una tarea determinada. En cambio, una relación "extender" o <<*extends*>>, indica opciones alternativas para un determinado caso de uso.

- **Diagrama de Clases**

Este tipo de diagrama es el más comúnmente usado, y la base primordial de toda solución orientada a objetos. Los diagramas de clases describen la estructura estática de un sistema determinado. Un rectángulo es el símbolo con el cual se representa una clase, y se divide en tres áreas.

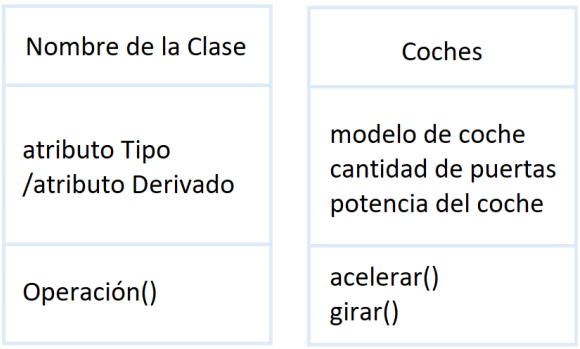


Figura 4.2 Notación de Clases en UML (fuente propia)

Una clase es una categoría o un grupo de cosas que tienen atributos (propiedades) y acciones similares. Un diagrama de clases está formado por varios rectángulos de este tipo conectados por líneas que representan las formas en que las diferentes clases se relacionan entre sí.

- **Diagrama de Actividades**

Este tipo de diagramas se utilizan para representar flujos de trabajo de negocios u operativos representados gráficamente para exponer la actividad de alguna parte del sistema o de alguno de sus componentes. Es decir, estos diagramas aclaran la naturaleza dinámica de un sistema a través del modelado del flujo sucediente de actividad en actividad.

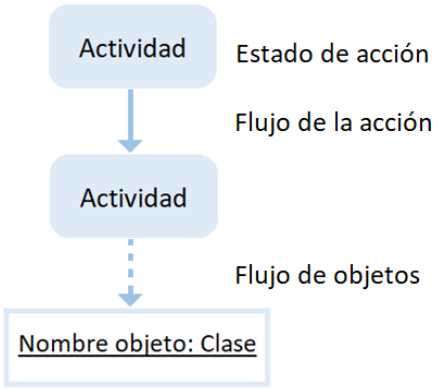


Figura 4.3 Notación de diagrama de actividades (fuente propia)

Tradicionalmente, los diagramas de actividad se utilizan para modelar el flujo de trabajo interno en una operación.

Los estados de acción representan las acciones que no son interrumpidas de los objetos Los flujos de las acciones, que son representados con flechas, enseñan las relaciones que existen entre los diferentes estados de acción. Por último, los flujos de objetos se refieren a la creación y o modificación de objetos a partir de actividades, es decir, significa que la acción está creando y o influyendo en dicho objeto. Los flujos de objetos son representados por fechas discontinuas.

5. Análisis del proyecto

El propósito de este apartado es ayudar al programador a conseguir una cierta comprensión de la naturaleza del problema. Una buena definición del problema es uno de los requisitos más importantes para llegar a una solución eficaz y conveniente.

Por ello es sumamente importante realizar una fase de análisis y diseño previa a la fase de implementación, y de esta manera evitar imprevistos y modificaciones que dificultan y ralentizan el desarrollo del proyecto.

5.1. Requisitos del proyecto

Un requisito es la condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. Estos requisitos deben ser claros y para poder documentarlos correctamente.

5.1.1. Requisitos funcionales

Un requisito funcional es el que define una función del sistema de software o a sus componentes.

Requisito funcional 1: Si al entrar en "Mapa" el GPS no está activo se debe tener la posibilidad de encenderlo.

Requisito funcional 2: El sistema debe mostrar la ubicación del usuario.

Requisito funcional 3: Se debe poder activar un mapa y comprobar el punto en el que se encuentra, el recorrido que se ha realizado hasta el momento y la distancia que queda hasta su destino.

Requisito funcional 4: Si se ha configurado "Solo conexión Wi-Fi" y no existe tal conexión no se mostrarán mapas de Google Maps debido al consumo de datos que tiene sobre la tarifa de datos.

Requisito funcional 5: Las distancias entre la localización del usuario y los diferentes restaurantes se muestran en metros si la distancia es inferior a 2 kilómetros, si la distancia es mayor se muestra en kilómetros.

5.1.2. Requisitos no funcionales

Requisito no funcional 1: Implementación de la aplicación únicamente para su uso en móviles.

Requisito no funcional 2: El dispositivo debe tener GPS, tenerlo activo y con una localización valida.

Requisito no funcional 3: El dispositivo tiene que tener conexión a internet para poder visualizar el recorrido sobre un mapa de Google.

Requisito no funcional 4: Desarrollar una interfaz con un diseño moderno y fácil de utilizar.

Requisito no funcional 5: El dispositivo tiene que tener una versión Android KitKat (4.4) o superior.

5.2. Casos de Uso del proyecto

Un *caso de uso* es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

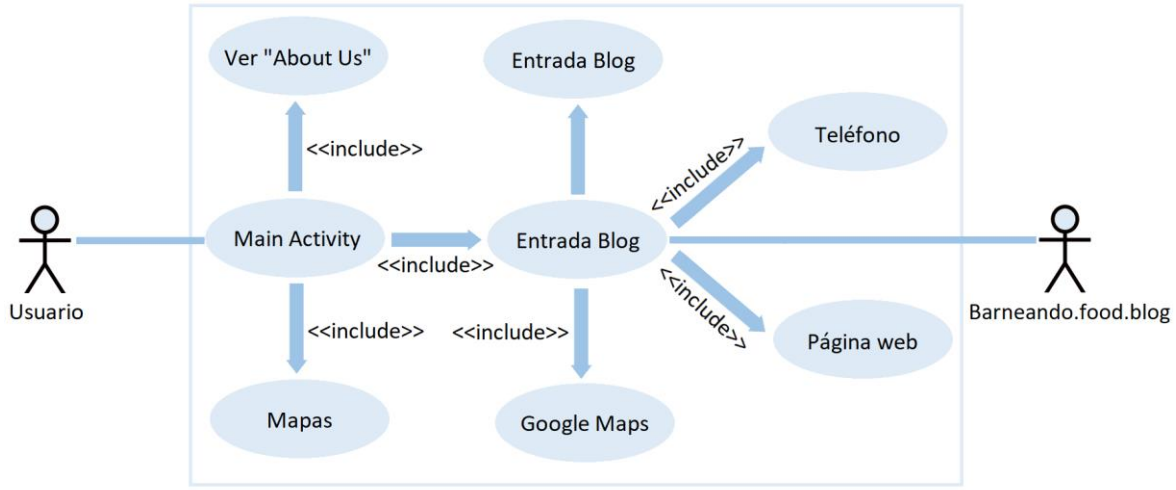


Figura 5.1 Diagrama casos de uso (fuente propia)

Como se puede ver en la ilustración anterior, el diagrama de casos de uso identifica los comportamientos ejecutantes de en este caso, un software. También explica con qué usuarios y/o entidades exteriores tiene interacción.

5.2.1. Casos de uso textuales

Para una mayor compresión del diagrama de casos de uso anterior a continuación, se especifica textualmente los casos de uso, describiéndolos cada uno de ellos por separado.

UC1	
Nombre: *Main Activity (menu principal)*	
Breve descripción	Este caso de uso el actor podrá ver una pequeña introducción de cada una de las entradas del blog y saber a qué distancia se encuentra de cada restaurante
Actores	Usuario
Flujo de eventos	Se inicia al entrar en la aplicación
Flujo principal	El actor abre la aplicación
Flujo alternativo	Sin flujo alternativo
Precondiciones	Sin precondiciones
Postcondiciones	Que el actor pueda ver satisfactoriamente una visión global de todas y cada una de las entradas del blog y escoger cual es la que quiere leer en ese momento.

Tabla 5.1 Caso de uso UC1

UC2	
Nombre: *About us*	
Breve descripción	Este caso de uso permitirá al actor ver información referente a los autores del proyecto y del blog *Barneando*
Actores	Usuario
Flujo de eventos	Se inicia cuando se pulsa el botón "*About us*"
Flujo principal	1. El actor pulsará el botón que permite ver la información mencionada anteriormente 2. Al pulsar el botón, se abrirá un dialogo que muestra la información sobre los autores de la aplicación y del blog *Barneando* 3. El actor podrá pulsar el botón "atrás" y volverá al menú principal
Flujo alternativo	Sin flujo alternativo
Precondiciones	Sin precondiciones
Postcondiciones	Que el actor pueda leer satisfactoriamente la información sobre los creadores de la aplicación y del blog *Barneando*.

Tabla 5.2 Caso de uso UC2

UC3	
Nombre: *Mapas*	
Breve descripción	Este caso de uso permitirá al actor ver todos los restaurantes o bares de las entradas del blog en un mapa.
Actores	Usuario

Flujo de eventos	Se inicia cuando se pulsa el botón “Mapa”
Flujo principal	1. El actor pulsará el botón que permite ver la información mencionada anteriormente 2. Al pulsar el botón, se abrirá una nueva actividad que muestra un mapa con la ubicación de todos los restaurantes, bares o cafeterías que salen en las entradas del blog *Barneando*. 3. El actor podrá pulsar el en cualquier punto fuera del dialogo y volverá a la actividad principal
Flujo alternativo	Sin flujo alternativo
Precondiciones	Sin precondiciones
Postcondiciones	Que el actor pueda ver y ubicar satisfactoriamente todos los restaurantes, bares o cafeterías en el mapa.

Tabla 5.3 Caso de uso UC3

UC4	
Nombre: *Entrada blog*	
Breve descripción	Este caso de uso permitirá al actor ver la entrada del blog.
Actores	Usuario
Flujo de eventos	Se inicia cuando se pulsa alguno de los botones de “leer más”
Flujo principal	1. El actor pulsará el botón que permite ver la información mencionada anteriormente 2. Al pulsar el botón, se abrirá una nueva actividad 3. En esta actividad se encuentra la descripción del restaurante, bar o cafetería en cuestión. 4. El autor puede pulsar el botón “teléfono” y se abrirá la aplicación teléfono para poder realizar la llamada 5. El autor puede pulsar el botón “dirección” y se abrirá la dirección correspondiente en Google Maps 6. El autor puede pulsar encima de la “página web” y se abrirá el navegador con la página web correspondiente 7. El autor puede valorar el restaurante 8. El autor podrá pulsar el botón “entrada siguiente” e ir a la entrada siguiente 9. El autor podrá pulsar el botón “entrada anterior” e ir a la entrada anterior 10. El actor podrá pulsar el botón “Home” y volverá al menú principal

Flujo alternativo	1. Si tras pulsar el botón "teléfono", es decir la acción del flujo básico 4, el usuario no ha dado permisos para usar la aplicación teléfono, se lanzará un *Toast* pidiendo los permisos. 2. Si tras pulsar el botón "dirección", es decir la acción del flujo básico 5, el usuario no ha dado permisos para usar la aplicación Google Maps, se lanzará un *Toast* pidiendo los permisos.
Precondiciones	Sin precondiciones
Postcondiciones	Que el actor pueda leer satisfactoriamente la información sobre la entrada del blog correspondiente, entrar en su página web de manera directa, llamar por teléfono, ver la ubicación en el mapa, valorar el restaurante e ir a la entrada anterior, a la siguiente o volver al menú principal.

Tabla 5.4 Caso de uso UC4

En la figura 4.2. se puede observar que como segundo actor se encuentra el blog original de *Barneando*. De este se deben extraer los datos partir de *rss* y *feed*. Este caso de uso no está implementado actualmente en la aplicación, pero se contempla para las mejoras posteriores de la aplicación.

6. Diseño de la aplicación

Después de haber establecido los diferentes requisitos del software, el diseño es la siguiente actividad técnica, seguida de la implementación y finalmente las pruebas. Cada una de estas actividades trasforma la información para finalmente obtener un software validado.

Para no construir un sistema inestable o un sistema que falle al realizar pequeños cambios, hay que poner hincapié en el diseño del software.

6.1. Diagramas de clases UML de la aplicación

Para tener una visión más completa de la aplicación, se han realizado los siguientes diagramas UML.

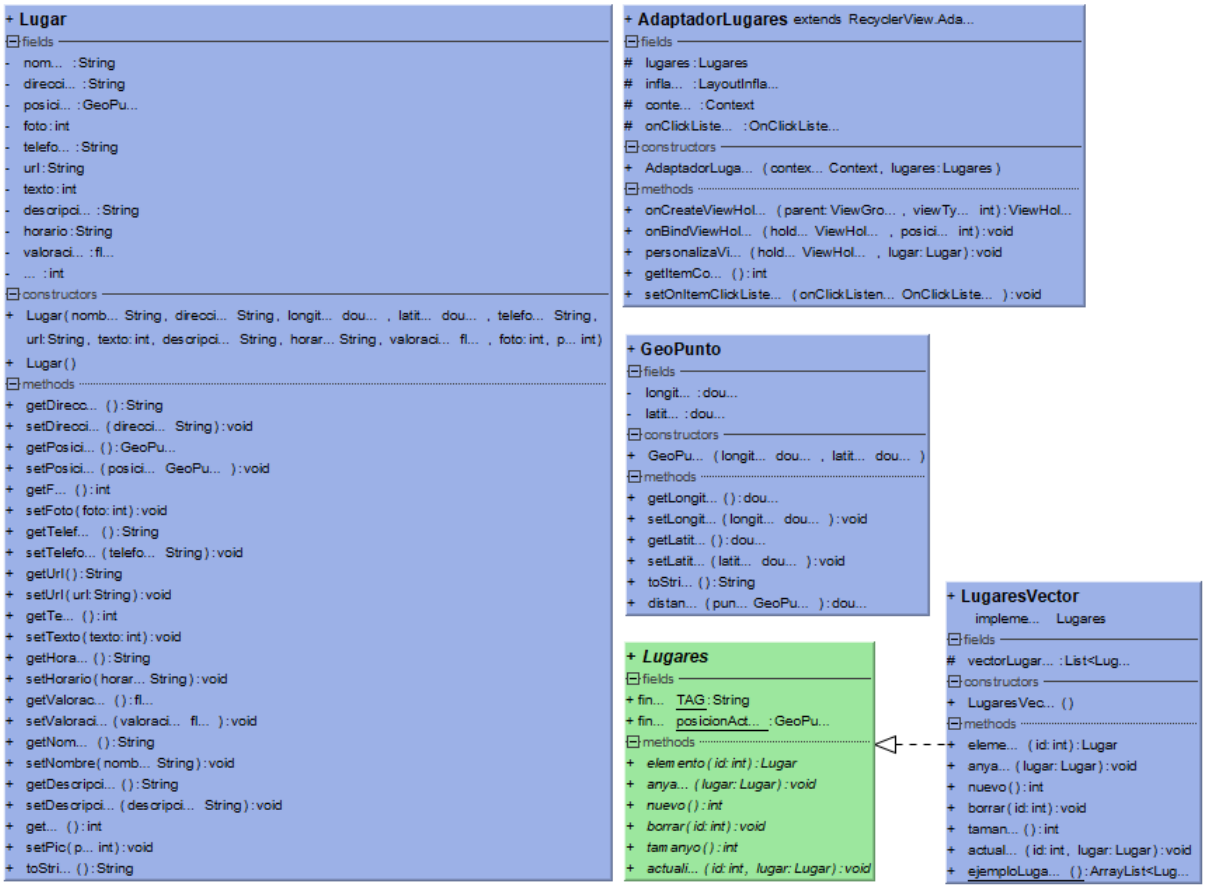


Figura 6.1 Diagrama UML de clases (fuente propia)

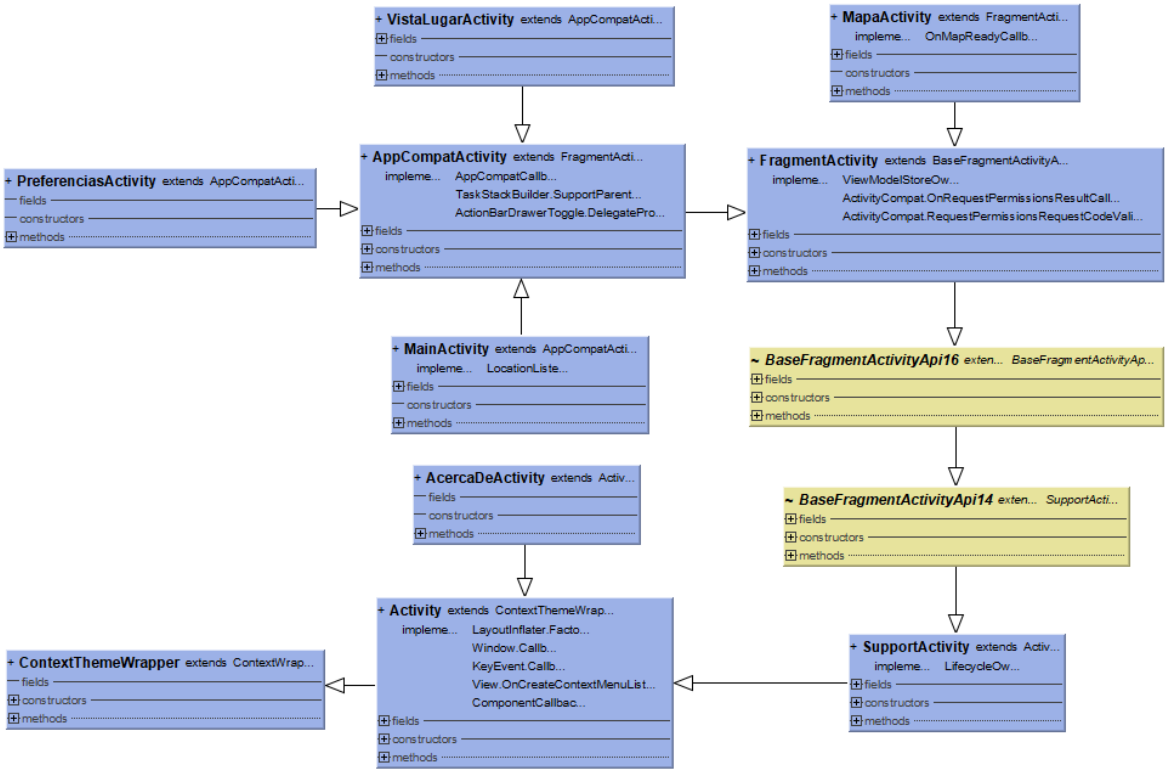


Figura 6.2 Diagrama UML de las clases Activity con solo extensiones (fuente propia)

Figura 6.3 Diagrama UML de las clases Activity (fuente propia)

6.2. Diseño de la interfaz gráfica

La interfaz es la parte más importante de la aplicación a ojos del usuario ya que es lo que puede ver y manipular para interaccionar con la misma. Esto hace que sea uno de los factores que más influyen en el éxito o fracaso de un producto, ya que si la apariencia es aburrida y poco bonita el usuario puede desecharla, no obstante, si tiene una estética intuitiva y agradable puede animar a ser usada.

Por esta razón se ha de tener especial dedicación con su diseño e implementación para así poder conseguir una apariencia bonita, clara y de fácil comprensión.

Para poder diseñar una buena interfaz gráfica hay que tener en los conceptos siguientes:

- **Usabilidad**: Se ha de implementar una aplicación fácil de usar y aprender.
- **Accesibilidad**: Se ha de asegurar de que cualquier usuario sea capaz de utilizar el producto sin mucha dificultad.
- **Solidez:** Los usuarios han de poder realizar sus objetivos sin problemas.
- **Consistencia**: Es una de las características más importantes y está muy relacionada a la usabilidad. Consiste en que, dentro de la aplicación, los mecanismos deben ser usados de la misma manera proporcionando la misma respuesta ante una misma acción. Ofreciendo una apariencia estética común.

A continuación, se muestra el diseño de cada una de las actividades que componen la aplicación y se realizara una pequeña explicación de las mismas.

- **Actividad principal (*Main Activity*):**

La actividad principal es la primera vista que el usuario obtiene nada más abrir la aplicación, por lo que es muy importante cuidar su estética y aportar la información de una forma organizada y clara.

Puesto que la aplicación, trata sobre restauración en la ciudad de Barcelona, la importancia de su funcionalidad reside en las diferentes entradas del blog y el servicio de localización que ofrece la aplicación. Por eso el contenido más útil y relevante que se puede mostrar en esta pantalla es un listado con las diferentes entradas del blog, a la distancia a la que cada bar o restaurante se encuentra del usuario en ese preciso momento. De esta manera el usuario puede acceder rápidamente a las diferentes entradas.

En la siguiente figura se muestra el diseño de esta primera pantalla en Android Studio.

Figura 6.4 Diseño pantalla principal (fuente propia)

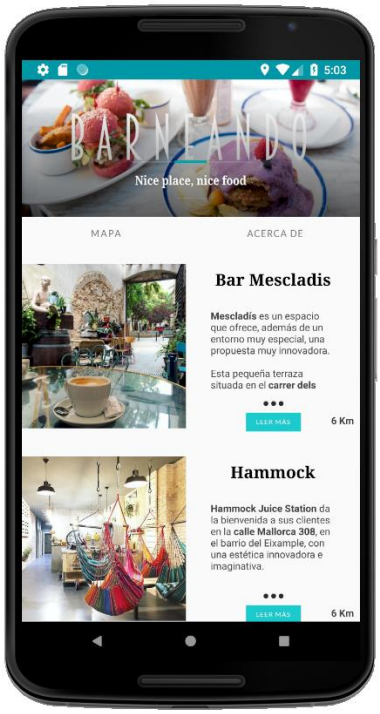


Figura 6.5 Pantalla principal (fuente propia)

Los elementos que componen esta pantalla principal son: una imagen, dos botones (mapa y acerca de) y una lista de las diferentes entradas del blog.

- **Actividad "acerca de "**

Figura 6.6 Diseño pantalla acerca de (fuente propia)

Figura 6.7 Pantalla "acerca de" (fuente propia)

Al pulsar el botón "Acerca de" se abre un dialogo que muestra la información sobre los autores de la aplicación y del blog *Barneando*.

- **Actividad "mapa"**

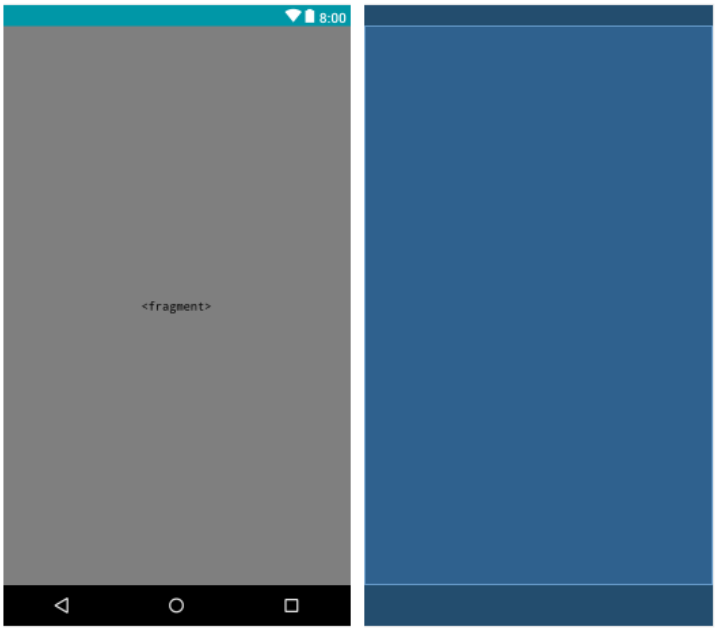


Figura 6.8 Diseño pantalla mapa (fuente propia)

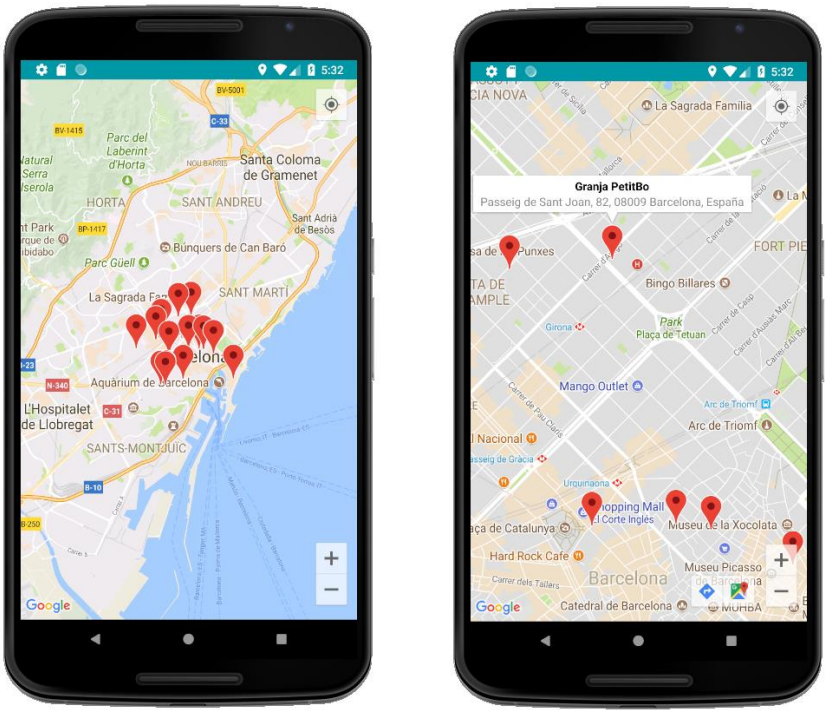


Figura 6.9 Pantalla mapa (fuente propia)

Al seleccionar el botón mapa se abre en Google Maps las localizaciones de todos los locales que están descritos en la aplicación.

- **Actividad "entrada "**

Figura 6.10 Diseño pantalla entrada 1 (fuente propia)

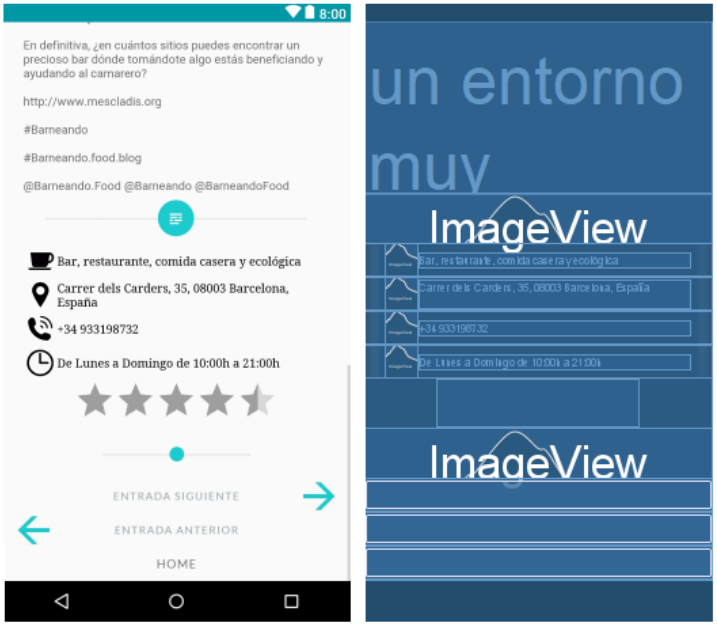


Figura 6.11 Diseño pantalla entrada 2 (fuente propia)

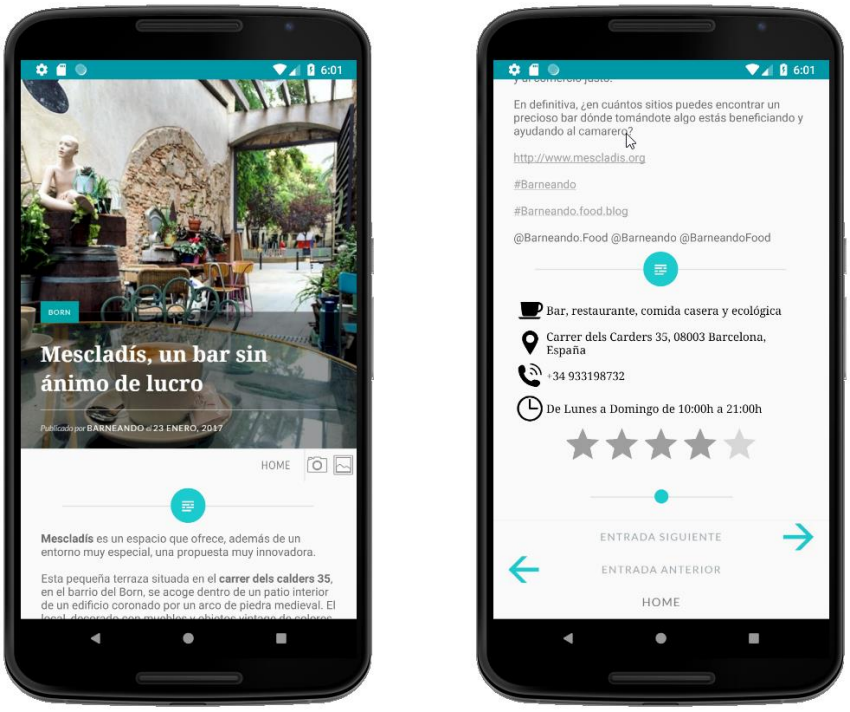


Figura 6.12 Pantalla entrada (fuente propia)

En las actividades de tipo entrada es donde se encuentra el mayor contenido de la aplicación. Cada una de ellas se compone por una serie de elementos:

El primer elemento que nos encontramos al abrir este tipo de actividades es con una foto emblemática del bar, restaurante o cafetería en cuestión, junto con su nombre y el barrio de Barcelona en el que está situado.

Seguidamente nos encontramos con un botón de home, por si se quisiera volver a la actividad principal.

A continuación, se puede leer una descripción sobre el local, donde se explica que es lo que lo hace especial. En este mismo texto, al final, se encuentran una serie de enlaces que redirigen hacia la página web del restaurante, el Instagram de *Barneando* y la página web del blog de *Barneando* respectivamente.

Más abajo se puede conocer las palabras claves de la descripción del local, su dirección (si se clica encima redirige a Google Maps), el número de teléfono (si se clica encima redirige a teléfono) y finalmente el horario de apertura.

Posteriormente existe una barra de valoración para que el usuario valore la experiencia.

Por ultimo y no menos importante, se encuentran tres botones; uno que permite al usuario ir a la entrada siguiente, uno que permite al usuario ir a la entrada anterior y el tercero vuelve a la actividad principal.

6.2.1. Definición de la paleta de colores de la aplicación

Cada aplicación ha de definir su propia paleta de colores que la diferencie de las demás. Para escoger la paleta de colores de la aplicación se ha tenido en totalmente en cuenta los colores que han sido utilizados en el blog original. De esta forma se ha intentado dar homogeneidad y continuidad, y no confundir al usuario.

Para escoger y definir los colores se ha utilizado la herramienta https://www.materialpalette.com/.

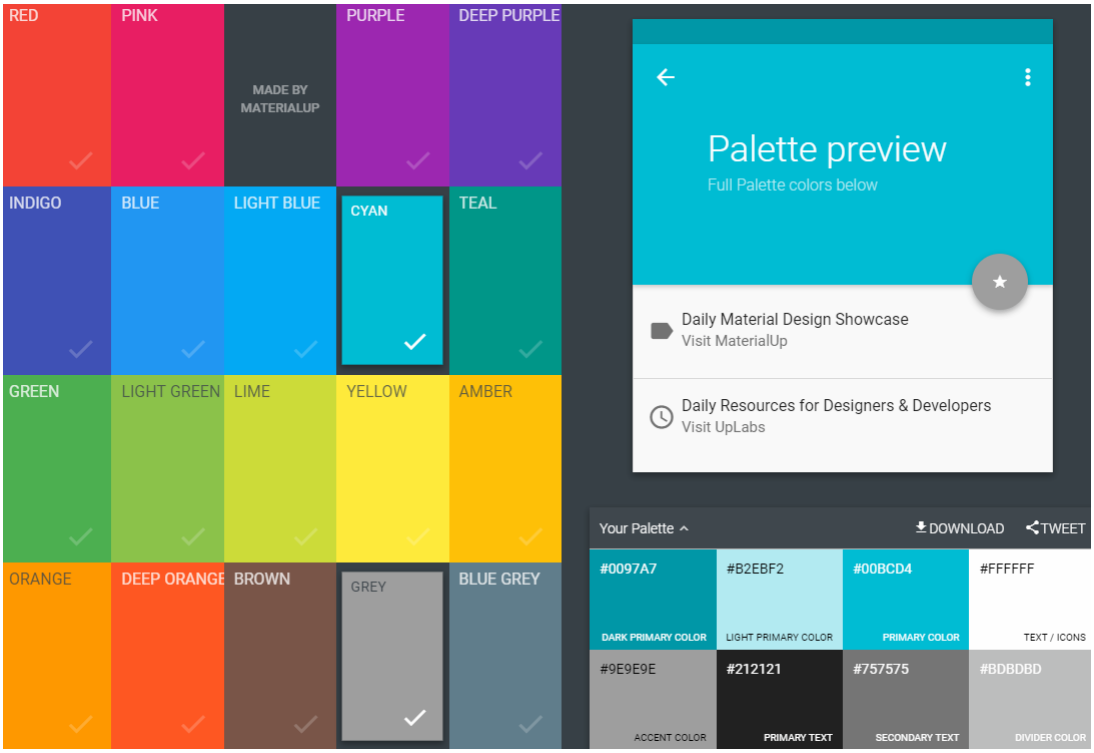


Figura 6.13 Paleta de colores de la aplicación (fuente: materialpalette.com)

6.2.2. Icono de la aplicación

A la hora de diseñar el icono de la aplicación, se ha de tener en cuenta algunos aspectos importantes. Primero de todo, Android ha sido creado para poder ser utilizado muchos dispositivos diferentes, eso quiere decir que existen una gran variedad de densidades gráficas. Por esa razón se ha de crear el icono en varias densidades gráficas.

En segundo lugar, también hay que tener en cuenta que la aplicación se ejecutara en un sistema donde son utilizadas guías de estilo.

Teniendo en cuenta estos factores se ha creado un icono sencillo, que cumple las guías de estilo de los dispositivos Android, y que es fácilmente relacionable con el blog de *Barneando*.

Se ha utilizado el color azul-turquesa característico del blog, junto con la B de *Barneando*.

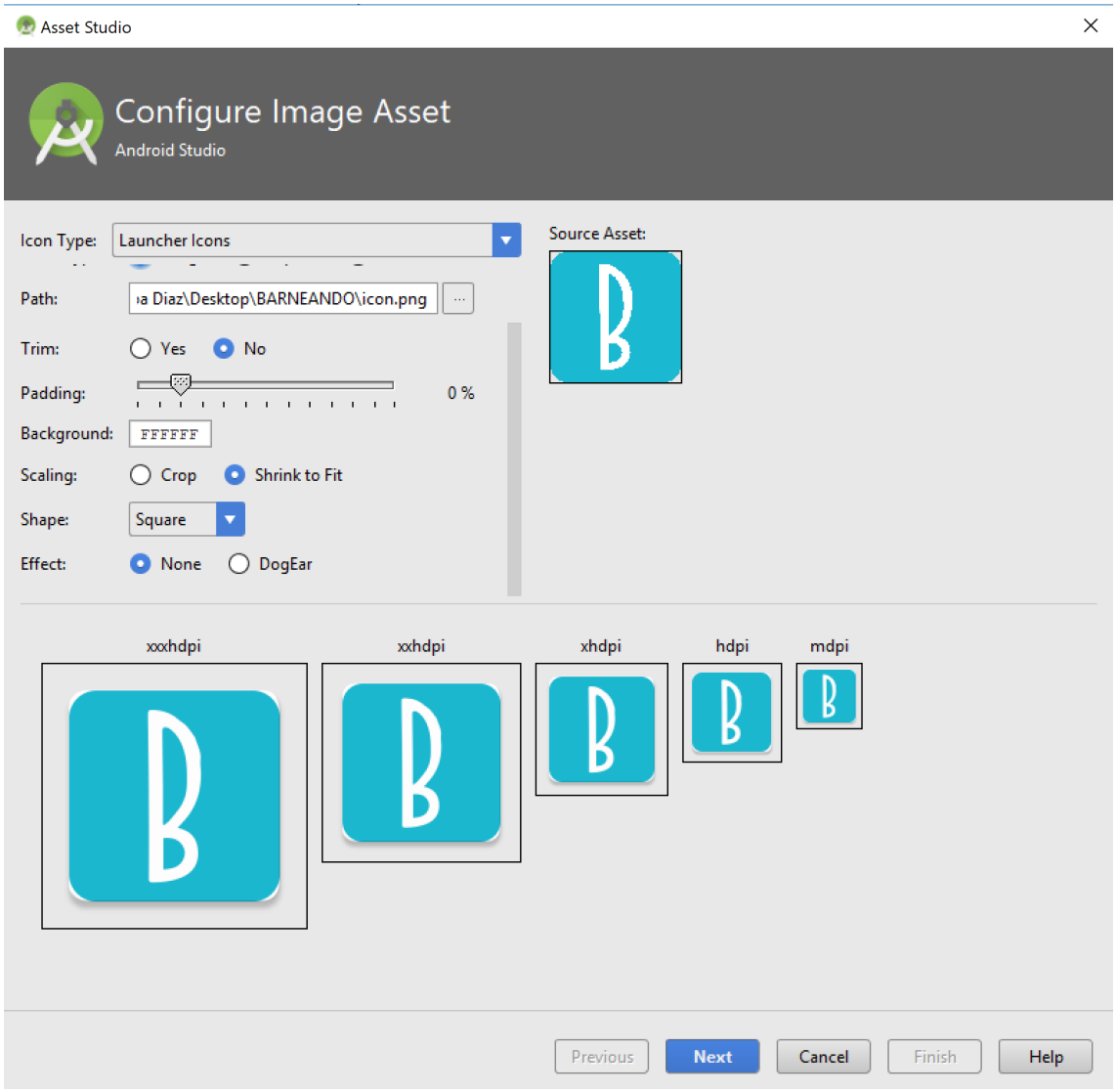


Figura 6.14 Icono de la aplicación Barneando (fuente propia)

7. Implementación

Una implementación o implantación es la realización de una aplicación, o la ejecución de un plan, idea, modelo científico, diseño, especificación, estándar, algoritmo o política.

En ciencias de la computación, una implementación es la realización de una especificación técnica o algoritmos como un programa, componente software, u otro sistema de cómputo. Por tanto, en esta fase del trabajo se ha obtenido el código de la aplicación a partir de la documentación obtenida durante las fases de análisis y diseño.

7.1. Herramientas

Como se ha comentado anteriormente, la aplicación ha sido desarrollada en Android Studio, utilizando el lenguaje de programación Java. Esta herramienta es el IDE oficial de Google para desarrollar aplicaciones en Android, y este ha sido el principal motivo para su elección en lugar de Eclipse.

La descarga de la herramienta se hace en http://developer.android.com/intl/sdk/index.html y es completamente gratuita, y disponible tanto para Windows como para Mac OS y Linux.

Además, la instalación indica que versión del *Java Development Kit (JDK)* es la óptima para la versión de Android Studio que hemos descargado. Ofrece un link para la descarga (también gratuita) desde la web de Oracle, ya que hay que descargarlo por separado.

7.2. Estructura proyecto Android

Los proyectos Android están divididos en diferentes carpetas, pero internamente se denominan paquetes, y es donde se encuentran todos los recursos del proyecto.

En estos paquetes hay unos con los ficheros de código java, otros con los archivos *xml* de visualización y configuración, y también otros recursos como las imágenes e iconos de la aplicación.

Cada proyecto formado con Android Studio está formado por cuatro partes:

- Descriptor de la aplicación (*manifests*)
- Código fuente en Java (*java*)
- Ficheros de recursos (*res*)
- Ficheros para construir el módulo (*Gradle Scripts*)

Cada elemento se almacena en una carpeta específica, que hemos indicado entre paréntesis.

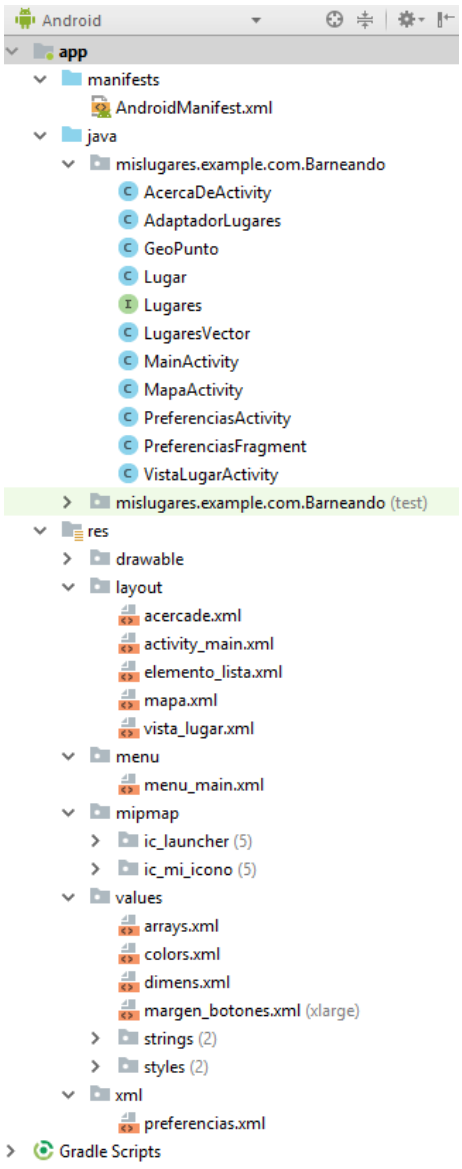


Figura 7.1 Estructura del proyecto Barneando en Android Studio (fuente propia)

En la figura 6.1 se pueden ver todos los *activity* y todas las clases utilizadas (en *mislugares.example.com.Barneando*), así como también los xml correspondientes a los *activity* (en la carpeta *layout*), a los menús (menu) y las configuraciones (*values*), y también se podían ver todas las imágenes utilizadas (*drawable*).

7.2.1. *Manifests (AndroidManifest.xml)*

En este fichero se describe el contenido que se encuentra en la aplicación Android y se definen:

- Nombre de la aplicación, los paquetes Java, icono de la aplicación, estilo de la aplicación, etc.
- El listado de componentes que forman la aplicación: Actividades, intenciones, servicios, proveedores de contenido y los receptores de anuncios
- Permisos que necesita la aplicación, la versión mínima del API de Android para poder ejecutarla, la versión de la aplicación, etc.
- Librerías vinculadas
- Características de hardware

7.2.2. *Java*

Esta carpeta es la que contiene el código fuente de la aplicación. Estos ficheros Java se almacenan en diferentes carpetas dependiendo del nombre de su paquete.

Por defecto, se crean tres clases, en tres carpetas diferentes:

- ***MainActivity:*** Clase de Java que contiene el código de la aplicación.
- ***ExampleInstrumentedTest:*** Clase de Java que se utiliza para realizar test de instrumentación, es decir, aquellos test que necesitan todo el sistema Android para probarse.
- ***ExampleUnitTest:*** Clase de Java que se utiliza para hacer test unitarios, es decir, aquellos test que no necesitan todo el sistema Android en funcionamiento para poder probarse.

7.2.3. *Res*

Esta es la carpeta que abarca todos aquellos recursos que son usados por la aplicación. Estos recursos son almacenados en varias subcarpetas dependiendo del tipo de recurso:

- ***drawable:*** Esta carpeta almacena los recursos que son de tipo imagen (en formato JPG o PNG) y los descriptores de imágenes XML.
- ***layout:*** Aquí se almacenan los ficheros XML que son vistas de la aplicación. Estas vistas permiten configurar las diversas pantallas que todas juntas constituirán la interfaz de usuario. El formato que se utiliza recuerda al HTML utilizado comúnmente en el diseño de páginas web.
- ***menu:*** En esta carpeta se encuentran los archivos XML con los menús de cada una de las actividades.
- ***mipmap:*** Esta carpeta tiene el mismo objetivo que la carpeta *drawable*, es decir, contiene ficheros de imágenes, pero en este caso son imágenes que no son escalables. A consecuencia, se debe definir la imagen en diferentes densidades graficas para que el sistema escoja la mejor en cada ocasión.

- ***values:*** Aquí se encuentran diferentes ficheros XML que indican valores usados en la aplicación. Esta carpeta se subdivide en cuatro: *Colors.xml* en la que se definen los diferentes colores de la aplicación, *Dimens.xml* en la que se definen los márgenes horizontal y vertical que se usan por defecto, *Strings.xml* donde se definen los diferentes textos que se encuentran en nuestra aplicación, pudiendo crear recursos alternativos que facilitan cambiar el idioma de la aplicación, y por último encontramos la subcarpeta *Styles.xml* donde se pueden definir el estilo y tema de la aplicación.
- ***xml:*** Carpeta donde se almacenan otros ficheros XML que son necesitados por la aplicación.

7.2.4. *Gradle Scripts*

Se trata de la carpeta en la que se almacenan unos determinados ficheros *Gradle*, los cuales permiten compilar y construir la aplicación. En este caso, Android Studio la utiliza como herramienta que se ocupa de compilar y construir el código.

7.3. Componentes de una aplicación

En este apartado del capítulo se van a describir de forma breve los elementos que componen la interfaz de usuario de una aplicación. Estos elementos son, por ejemplo, un botón, una entrada de texto, etc. Todas las vistas van a ser objetos descendientes de la clase View y, por tanto, pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero.

7.3.1. *Layout*

Se trata del agrupamiento de un conjunto de vistas de una determinada forma para construir una interfaz clara y ordenada. Existen diferentes tipos de *layouts* para organizar las vistas de distintas maneras; de forma lineal, en forma de cuadricula o indicando la posición absoluta de cada una de las vistas.

7.3.2. *Activity*

Una aplicación de Android, va a estar formada por un conjunto de elementos básicos de visualización, es decir, lo que se conoce coloquialmente como “pantallas de una aplicación”. Su función principal es crear la interfaz de usuario, la cual suele tener varias actividades independientes. Toda *Activity* ha de pertenecer a una clase descendiente de *Activity*.

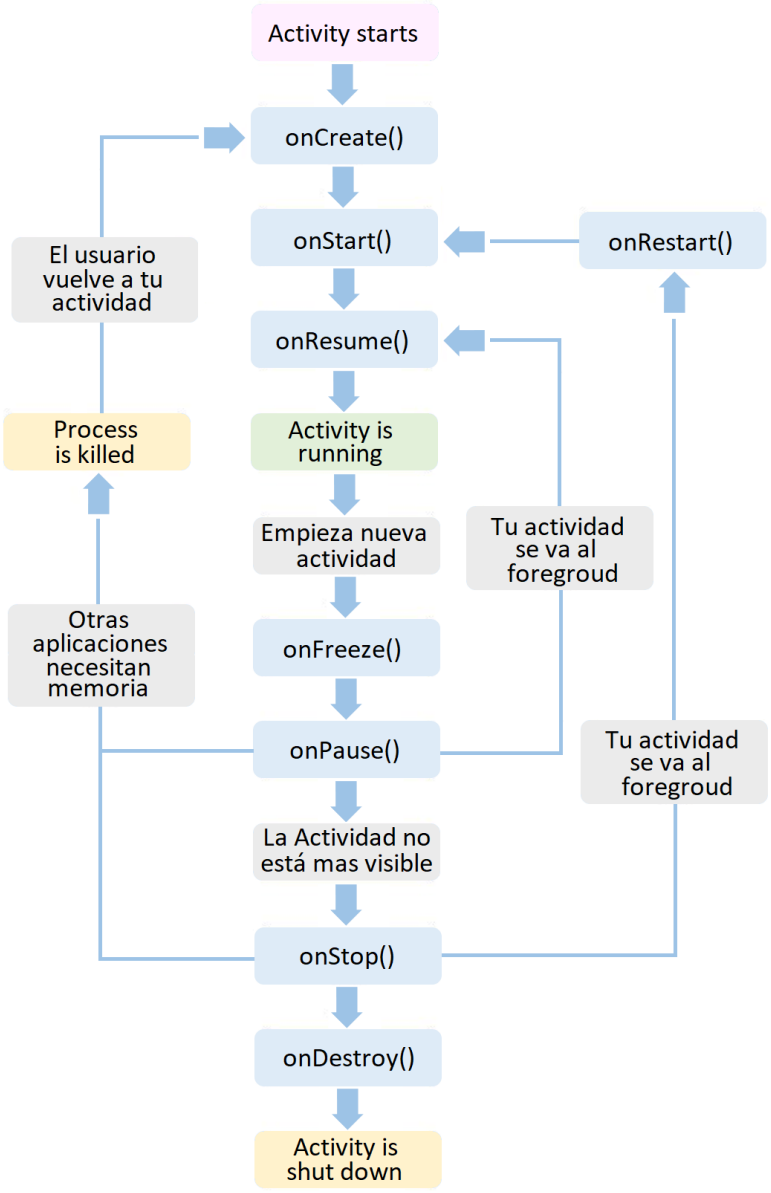


Ilustración 7.1 Ciclo de vida de una actividad (fuente propia)

Como se puede apreciar en la figura anterior, Android es sensible al ciclo de vida de una *Activity*, por lo tanto, es muy importante comprender y manejar los eventos relacionados con el ciclo de vida si se quiere crear aplicaciones estables.

Cada vez que una *Activity* cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. Así habrá que atender a los siguientes momentos:

- **OnCreate(Bundle):** Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de

datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

- **OnStart():** Nos indica que la actividad está a punto de ser mostrada al usuario.
- **OnResume():** Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- **OnPause**(): Indica que la actividad está a punto de ser lanzada a segundo plano (*foreground*), normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- **OnStop**(): La actividad ya no va a ser visible para el usuario. Hay que tener en cuenta que, si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- **OnRestart**(): Indica que la actividad va a volver a ser representada después de haber pasado por *onStop().*
- **OnDestroy**(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al *métodofinish().* Hay que tener en cuenta que, si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Como se ha podido observar, el ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros sistemas operativos. La mayor diferencia es que, en Android el ciclo de vida es controlado por el sistema, en lugar de ser controlado directamente por el usuario.

7.3.3. *Service*

Son aquellos procesos que se ejecutan en segundo plano (es algo parecido a un demonio en Unix o a un servicio de Windows) sin ninguna necesidad de interacción con el usuario.

En Android disponemos de dos tipos de servicios: servicios locales, que pueden ser utilizados por aplicaciones del mismo terminal y servicios remotos, que pueden ser utilizados desde otros terminales.

7.3.4. *Fragment*

Un *fragment* se podría describir como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades.

7.3.5. *Intent*

Representa la voluntad de lanzar alguna acción, como por ejemplo una llamada telefónica, acceder a mapas, ect. Se utiliza siempre y cuando se quiera lanzar una actividad o un servicio, lanzar un anuncio de tipo broadcast o comunicarnos con un servicio.

7.3.6. Broadcast Receiver

Sistema que Android utiliza para recibir y reaccionar frente a situaciones concretas, como por ejemplo cuando al dispositivo le queda poca batería o si este mismo recibe un mensaje.

7.3.7. Content Provider

Permite que se comparta información entre aplicaciones, como acceder a los contactos, a la galería de tu dispositivo, etc.

8. Análisis del impacto medioambiental

El impacto medioambiental de la aplicación informática sobre el blog *Barneando* es muy positivo. Este hecho no viene dado por la creación del software en sí, sino por el contenido de la aplicación.

En *Barneando*, casi todos los bares, restaurantes y cafeterías que se describen, tanto en la aplicación como en el blog, aportan su granito de arena en este tema tan crucial hoy en día.

Algunos bares cumplen una labor social, como el **bar Mescladís**, que ha creado una innovadora metodología de desarrollo que ayuda tanto a colectivos inmigrantes como autóctonos con dificultades a integrarse y a encontrar trabajo en algunos de sus centros, para lograr una mejor inserción social y laboral.

L'antic teatre, es otro ejemplo: este bar es la principal fuente de financiamiento del centro social y cultural en el que se encuentra**;** los ingresos que se generan se destinan en su totalidad a la actividad del recinto para mantener todos los espacios, sus infraestructuras y su actividad completa.

Otros utilizan solamente productos orgánicos, y batidos naturales como el **Bar Jammock**, o el **Café Cometa.**

El **Mercader de l'Eixample** es el perfecto ejemplo de comida "*slow food".* En este restaurante encontrado en el corazón de Barcelona, se estandariza los conceptos y sabores de la cocina apostando por productos Km0 (de proximidad) y ecológicos. Ofrece únicamente productos de temporada, por eso los platos van variando.

Otro ejemplo es **The Juice House**, otro bar – cafetería – restaurante**,** con proveedores de proximidad y productos de calidad; su pan es de masa madre y artesanal, lo hace Enric de *Pa Solá* a las afueras de Barcelona, todos los productos lácteos (*Granja el Prat*) y sus huevos provienen de animales que viven felices y en libertad y la *Bodega familiar Ros Marina* les trae una selección de vinos ecológicos destilados en el Penedés.

Estos son solo unos pocos restaurantes que puedes encontrar en la aplicación, y cada uno de ellos es especial por alguna cosa.

9. Conclusiones

Antes de emprender este proyecto, no se disponía de ningún conocimiento sobre la programación Java o la plataforma Android Studio. Una vez asentada la base teórica sobre qué era Java, la programación orientada a objetos y en qué consistía la herramienta Android, se manifestaron otros conceptos que se debían considerar a la hora de realizar una aplicación móvil.

A priori, se ha tenido que estudiar, conocer y preparar el entorno software, instalando Android Studio y Eclipse. Asimismo, se ha tenido que configurar para poder hacer uso del SDK (Kit de Desarrollo de Software) de Android, que permite programar y depurar aplicaciones. Seguidamente, se ha realizado un análisis con los diferentes requisitos y los casos de uso del proyecto, así como la elaboración del diseño y finalmente, la implementación.

Al desarrollar la aplicación se ha conseguido afianzar los conocimientos teóricos. Se ha entendido como se programa en Android, como utilizar el API de Google Maps y el posicionamiento por GPS. Además, al programarse la gran mayoría en lenguaje Java, se ha aprendido uno de los lenguajes de programación más utilizados hoy en día.

La aplicación desarrollada se ha validado en diferentes dispositivos reales, como en un Xiaomi Mi X4 y en un Motorola MotoG de segunda generación. También se ha validado y probado en una gran variedad de dispositivos emulados, con diversas resoluciones de pantalla y API. Así, se ha podido corroborar que la interfaz se adapta una gran cantidad de dispositivos sin problema.

De esta forma, he conseguido familiarizarme con los componentes que forman una aplicación, las diferentes versiones existentes y el ciclo de vida de una actividad para así poder crear apps estables y compatibles con el mayor número de dispositivos.

Hoy en día, encontramos en el mercado aplicaciones parecidas a la creada. El objetivo nunca ha sido inventar algo nuevo, sino aprender a realizar una aplicación sencilla con una gran funcionalidad. La realización de *Barneando* me ha permitido aprender una nueva rama de la ingeniería en la que me siento cómoda y curiosa por saber más.

Con todo, puedo concluir que se han logrado la totalidad de los objetivos mencionados al empezar este proyecto.

10. Presupuesto

10.1. Presupuesto informático

La tabla que se encuentra a continuación, recoge los elementos informáticos necesarios para la realización de este proyecto juntamente con sus respectivos costos.

Concepto	Unidades	Coste unitario (€)	Coste total (€)
Ordenadores			
Ordenador Portátil ASUS VivoBook Pro 15	1	1.149	1.149
Ordenador Portátil Microsoft Surface	1	1.148,99	1.148,99
Teléfonos móviles			
Lenovo ZUK Z2 Pro	1	244,88	244,88
Motorola Moto g (3a Generación)	1	210	210
Recursos Software			
Microsoft Office	1	149	149
Android Studio	1	0	0
Otros			
Monitor LG 22M47VQ-P	1	109,44	109,44
Teclado y ratón inalámbrico Microsoft Wireless Desktop 3000	1	84,52	84,52
Disco Duro exterior CORE	1	65	65
		TOTAL	**3.160,83**

Tabla 10.1 Presupuesto de los elementos informáticos utilizados en el proyecto (fuente propia)

10.2. Presupuesto de mano de obra

La tabla que se encuentra a continuación, recoge el costo por mano de obra del personal en función de las horas dedicadas a cada tipo de trabajo.

El precio por hora se ha calculado teniendo en cuenta el suelto base de un programador Junior en Barcelona (22.000€ brutos anuales).

Concepto	Horas (h)	Coste unitario (€)	Coste total (€)
Investigación	100	10	1.000
Diseño e ingeniería	300	12	3.600
Redacción de la memoria	180	8	1.440
		TOTAL	**6.040**

Tabla 10.2 Presupuesto del personal en función a las horas dedicadas a cada trabajo (fuente propia)

10.3. Presupuesto de material de oficina

La tabla que se encuentra a continuación, recoge el costo correspondiente del material de oficina y impresión del proyecto.

Concepto	Unidades	Coste unitario (€)	Coste total (€)
Paquete de folios A4	1	4,75	4,75
DVD	2	1	2
Caja de proyectos	1	5	5
Impresión del proyecto	2	50	100
Encuadernación del proyecto	2	2,5	5
		TOTAL	**116,75**

Tabla 10.3 Presupuesto del material de oficina (fuente propia)

10.4. Presupuesto total

La tabla que se encuentra a continuación, recoge el costo total del presente proyecto.

Concepto	Coste total (€)
Presupuesto informático total	3.160,83
Presupuesto de mano de obra total	6.040
Presupuesto de material de oficina total	116,75
TOTAL	**9.317,58**

Tabla 10.4 Presupuesto total del presente proyecto (fuente propia)

Bibliografía

Academiaandroid.com. (2018). Academia Android – Formación online para creadores de Apps. [online] Available at: https://academiaandroid.com/.

Androidcurso.com. (2018). Máster en Desarrollo de Aplicaciones Android. [online] Available at: http://www.androidcurso.com.

Android Developers. (2018). Build for Android. [online] Available at: https://developer.android.com.

Ardións, A. (2018). Tutoriales Android Studio. [online] Android Studio FAQs. Available at: https://androidstudiofaqs.com.

CódigoJavaLibre. (2018). Código Java Libre. [online] Available at: http://www.codigojavalibre.com/.

Discover.deviceatlas.com. (2017). The Mobile Web Intelligence Report Q2. [online] Available at: https://discover.deviceatlas.com/mobile-web-intelligence-report-q2-2017/.

Eclipse Foundation, I. (2018). The Platform for Open Innovation and Collaboration | The Eclipse Foundation. [online] Eclipse.org. Available at: https://eclipse.org/.

Google Developers. (2018). Google Maps APIs | Google Developers. [online] Available at: https://developers.google.com/maps.

Material Palette. (2018). Material Design Color Palette Generator. [online] Available at: https://www.materialpalette.com/.

Mktefa.ditrendia.es. (2017). Informe Mobile en España y en el Mundo. [online] Available at: http://mktefa.ditrendia.es/hubfs/Ditrendia-Informe%20Mobile%20en%20Espa%C3%B1a%20y%20en%20el%20Mundo%202017.pdf.

Mundojava.net. (2018). Introducción a la programación orientada a objetos | Curso de Introducción a Java. [online] Available at: http://www.mundojava.net/introduccion-a-la-programacion-orientada-a-objetos.html?Pg=java_inicial_4_2.html.

sgoliver.net. (2018). Interfaz de usuario en Android: Fragments. [online] Available at: http://www.sgoliver.net/blog/fragments-en-android/.

Sites.google.com. (2018). Arquitectura Android - Software de Comunicaciones. [online] Available at: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android.

Sites.google.com. (2018). Encapsulamiento y visibilidad en Java - Apuntes Android. [online] Available at: https://sites.google.com/site/pruebajoseog/practicas/7-encapsulamiento-y-visibilidad-en-java.

Statista.com. (2018). Statista - The Statistics Portal for Market Data, Market Research and Market Studies. [online] Available at: https://statista.com.

Teatroabadia.com. (2018). Programas del UML. [online] Available at: http://www.teatroabadia.com/es/uploads/documentos/iagramas_del_uml.pdf.

TERMINALES ANDROID. (2018). Developers. [online] Available at: http://www.terminalesandroid.com/.

Tomás Gironés, J. (2016). El gran libro de Android. 5ª ed. Marcombo.

Tomás Gironés, J. (2017). Course edX - Introducción a la Programación. [online] Courses.edx.org. Available at: https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+1T2017/course/